

UNIVERSIDAD DE ALCALÁ DE HENARES
Escuela Politécnica

**ESTUDIOS COMPLEMENTARIOS A LA TITULACIÓN EN
INGENIERÍA EN SISTEMAS DE INFORMACIÓN DE LA
UNAN-LEÓN**

PROYECTO DE FIN DE CARRERA



**PLAN DOCENTE PARA LA ASIGNATURA DE
*ADMINISTRACIÓN DE SISTEMAS OPERATIVOS***

JULIO CÉSAR GONZÁLEZ MORENO

**ALCALÁ DE HENARES, ESPAÑA
JUNIO 2008**

UNIVERSIDAD DE ALCALÁ DE HENARES
Escuela Politécnica

**ESTUDIOS COMPLEMENTARIOS A LA TITULACIÓN EN
INGENIERÍA EN SISTEMAS DE INFORMACIÓN DE LA
UNAN-LEÓN**

PROYECTO DE FIN DE CARRERA

**PLAN DOCENTE PARA LA ASIGNATURA DE
*ADMINISTRACIÓN DE SISTEMAS OPERATIVOS***

Autor: Julio César González Moreno
Directores: Elena Campo Montalvo
David Fernández Barrero

TRIBUNAL

Presidente: Francisco Javier Ceballos Sierra.

Vocal 1: Elena Campo Montalvo.

Vocal 2: David Fernández Barrero.

Calificación: _____

Fecha: 20 de junio de 2008

Dedicatoria y Agradecimiento

A mi familia, por haberme dado su apoyo incondicional antes de mí llegada a España y cuando en algunos malos momentos me sentía triste,

A mis amigos, por llevarme siempre en sus pensamientos deseándome lo mejor durante toda mi estancia,

A todas las personas que han hecho posible que varios Nicaragüenses tengamos la oportunidad de poder ampliar y desarrollar nuestros conocimientos, a través de este intercambio académico, técnico y cultural.

ÍNDICE GENERAL

I. INTRODUCCIÓN	21
II. OBJETIVOS	22
III. SITUACIÓN ACTUAL DE LA ASIGNATURA EN EL PÉNSUL ACADÉMICO DE INGENIERÍA EN TELEMÁTICA	23
IV. RELACIÓN CON OTRAS ASIGNATURAS	24
V. CONTENIDO DEL TEMARIO.....	25
VI. METODOLOGÍA DIDÁCTICA Y MATERIAL DIDÁCTICO A UTILIZAR.....	27
VII. PLANIFICACIÓN TEMPORAL	28
VIII. METODOLOGÍA DE EVALUACIÓN	29
IX. DESARROLLO DEL TEMARIO	31
<i>TEMA 0: PRESENTACIÓN DE LA ASIGNATURA.....</i>	33
Presentación del docente	33
Información de la asignatura	33
Objetivos de la Asignatura.....	33
Temario.....	33
Material de estudio	33
Metodología de evaluación	34
Tutorías.....	34
<i>TEMA 1: INTRODUCCIÓN</i>	35
1. Repaso de conceptos importantes sobre sistemas operativos	37
1.1. ¿Qué es un sistema operativo?	37
1.2. Concepto de proceso.....	37
1.3. Concepto de núcleo.....	37
1.3.1. Tipos de núcleo	38
1.4. Conceptos de: Multitarea, multiusuario y multiplataforma.....	40
Multitarea	40
Multiusuario.....	40
Multiplataforma.....	40
2. Introducción a Unix.....	41
2.1. Historia de Unix	41
2.2. Aparición de Linux	43
2.3. Distribuciones de Linux	44
2.3.1. Principales distribuciones	45
3. Software libre y Linux	49
3.1. GNU	49
3.2. Libertad y coste	50
3.3. Open Source	51
3.4. Licencias en el software libre	52
3.4.1. Licencias tipo BSD	52
3.4.2. La licencia Pública General de GNU (GNU GPL)	53
3.5. Licencias de otros recursos libres.....	53

3.5.1. Licencia de documentación libre de <i>GNU</i>	53
3.5.2. Licencias de <i>Creative Commons</i>	54
4. Introducción a la administración	56
4.1. Ciclo de vida del sistema	56
Análisis de requisitos del sistema	56
Diseño del sistema	56
Implantación del sistema	56
Configuración hardware y software de forma que el sistema cumpla los requisitos exigidos ..	57
Administración y mantenimiento (explotación)	57
Migración, desmantelamiento del sistema	57
4.2. El administrador del sistema	57
4.3. La primera regla del administrador	58
4.4. Responsabilidades del administrador	58
Responsabilidades hardware	58
Responsabilidad software	59
Responsabilidades derivadas del software del sistema	59
Responsabilidades derivadas del software específico	59
Responsabilidades sobre los usuarios	59
Aspectos éticos de la administración de sistemas	60
4.5. Seguridad en la administración	60
TEMA 2: INSTALACIÓN BÁSICA	63
1. Conceptos necesarios previos a la instalación	64
1.1. Concepto de sistema de archivos	64
1.2. Concepto de <i>montaje</i> de sistema de archivos	65
1.3. Concepto de <i>live CD</i>	65
1.4. Concepto de partición de disco	66
1.5. Concepto de espacio de <i>intercambio</i>	66
1.6. Concepto de MBR (Master Boot Record)	66
1.7. Concepto de gestor de arranque	67
1.8. Conceptos relacionados al software	67
2. Tareas de preparación para la instalación de Linux	68
2.1. Visión general de la instalación	68
2.2. Repartición del disco o discos duros	69
2.3. Requerimientos de la partición de Linux	70
2.4. Creación del espacio de <i>intercambio</i>	76
2.5. Creación del sistema de archivos	77
2.6. Instalar el software	77
2.7. Instalar el gestor de arranque <i>GRUB</i>	79
3. Procedimientos posteriores a la instalación	79
3.1. Creación de una cuenta de usuario no <i>root</i>	79
3.2. Pedirle ayuda a nuestro sistema	80
3.3. Edición del archivo <i>/etc/fstab</i>	80
3.4. Cerrar el sistema	81
TEMA 3: INSTALACIÓN, ACTUALIZACIÓN Y COMPILACIÓN DE PROGRAMAS	83
1. Actualización de software	84
2. Procedimiento general de actualización	85
3. Sistema de gestión de paquetes <i>RPM</i>	86
4. Sistemas de gestión de paquetes <i>.deb</i>	89
4.1. Utilizar <i>dpkg</i>	89

4.2. Utilizar <i>apt</i>	92
4.2.1. El corazón de <i>apt</i>	92
4.2.2. Tipos de paquetes según su prioridad	93
4.2.3. Grado de dependencia entre paquetes	93
4.2.4. Acciones sobre paquetes	94
4.2.5. Estado de instalación de los paquetes	94
4.2.6. Utilizar <i>apt-cache</i>	95
4.2.7. Utilizar <i>apt-get</i>	95
5. Software no proporcionado en paquetes	96
5.1. Actualización de bibliotecas	99
TEMA 4: INICIO Y CIERRE DEL SISTEMA.....	105
1. Inicio del sistema	106
1.1. Disquete de arranque.....	106
1.2. El gestor de arranque <i>GRUB</i>	108
1.2.1. El archivo <i>/etc/grub.conf</i>	109
1.2.2. Especificar las opciones del arranque	111
1.2.3. Eliminar <i>GRUB</i>	112
2. Inicio e inicialización del sistema	112
2.1. Mensajes de inicio de núcleo	112
2.2. Archivos <i>init</i> , <i>inittab</i> y <i>rc</i>	114
2.3. Archivos <i>rc</i>	116
3. Modo de un solo usuario	117
4. Cierre del sistema	118
TEMA 5: ADMINISTRACIÓN DE USUARIOS, GRUPOS Y PERMISOS.....	121
1. Tipos de cuentas de usuario	122
1.1. Cuenta de usuario completa o cuenta <i>shell</i>	122
1.2. Cuenta de acceso restringido	122
1.3. Programas y <i>daemons</i>	123
2. Usuarios y <i>uids</i>	123
3. Pseudo-conexiones.....	123
4. Gestión de usuarios y grupos.....	123
4.1. El archivo <i>passwd</i>	124
4.2. Contraseñas ocultas	125
4.3. El archivo de grupo.....	127
4.4. El archivo <i>gshadow</i>	129
5. Operaciones con las cuentas de usuario	130
5.1. Crear	130
5.2. Eliminar.....	132
5.3. Deshabilitar	132
5.4. Modificar	133
6. Otro método de autenticación: <i>PAM</i>	134
6.1. ¿Qué es <i>PAM</i> ?	134
6.2. Grupos de gestión	135
6.3. Arquitectura	136
TEMA 6: EL SISTEMA DE ARCHIVOS.....	137
1. Concepto de archivo y de sistema de archivos.....	139
2. Los <i>inodos</i>	140
3. El superbloque	141

4. El sistema de archivos <i>ext2</i>	142
5. El sistema de archivos <i>ext3</i>	143
5.1. ¿Cómo surge <i>ext3</i> ?	143
5.2. ¿Qué es <i>journaling</i> ?	143
6. El estándar de jerarquía del sistema de archivos	144
7. Algunos directorios interesantes	146
El directorio raíz “/”	146
Directorio <i>/bin</i>	146
Directorio <i>/sbin</i>	146
Directorio <i>/boot</i>	146
Directorio <i>/dev</i>	147
Directorio <i>/etc</i>	147
Directorios <i>/home</i> y <i>/root</i>	148
Directorio <i>/lib</i>	148
Directorio <i>/usr</i>	148
Directorio <i>/var</i>	148
Directorio <i>/mnt</i>	149
Directorio <i>/opt</i>	149
Directorio <i>/lost+found</i>	149
Directorio <i>/proc</i>	149
Directorio <i>/tmp</i>	149
8. Nombres de archivos y directorios	149
8.1 Convenios en los nombres de los archivos	150
9. Tipos de archivos	150
9.1. Archivos normales	151
9.2. Archivos de directorio	151
9.3. Directorios y discos físicos	151
9.4. Enlaces	152
9.5. Archivos especiales	153
10. Atributos existentes en el sistema de archivos de Linux	154
11. Propiedad y permisos de los archivos	156
11.1. ¿Qué significan los permisos?	156
11.2. Propietarios y grupos	157
12. Puntos adicionales del sistema de archivos	158
12.1. Máscara frente a umáscara	158
12.2. Establecer ID de usuario y de grupo (<i>SUID</i> y <i>SGID</i>)	159
12.3. El Sticky bit o bit adhesivo	159
13. Sistemas de archivos distribuidos (<i>DFS</i>)	160
13.1. <i>SAMBA</i>	161
13.1.1. Evolución histórica de <i>SAMBA</i>	161
13.1.2. Servicios proporcionados por <i>SAMBA</i>	162
13.2. Sistema de archivos en red de <i>Sun Microsystems: NFS</i>	163
13.2.1. Beneficios proporcionados por <i>NFS</i>	164
13.3. Sistema de información de redes de <i>Sun Microsystems: NIS</i>	164
13.4. Integrando <i>NIS</i> y <i>NFS</i>	165
TEMA 7: SISTEMAS DE RED	167
1. Introducción a TCP/IP	169
2. Servicios sobre TCP/IP	169
3. ¿Qué es TCP/IP?	170

4. La pila de protocolos IP	172
5. Dispositivos físicos (hardware) de red	172
6. Conceptos TCP/IP.....	174
7. Direcciones TCP/IP	177
8. Componentes de la dirección de red	178
8.1. Dirección de la máscara de red	178
8.2. Dirección de red	179
8.3. Dirección de puerta de acceso	179
8.4. Dirección de difusión	179
9. Configurar la red	179
9.1. Configuración de la interfaz <i>NIC (Network Interface Controller)</i>	179
9.2. Configuración del <i>Name Resolver</i>	180
9.3. Configuración del encaminamiento	182
9.4. Configuración del <i>inetd</i>	183
9.5. Configuración adicional: <i>/etc/protocols</i> y <i>/etc/networks</i>	186
9.6. Algunos aspectos de seguridad a tomar en cuenta.....	187
10. Aplicaciones seguras	187
10.1. Secure Shell (<i>ssh</i>).....	187
10.1.1. Características del protocolo <i>ssh</i>	188
10.1.2. ¿Por qué usar <i>ssh</i> ?	189
10.1.3. Ejemplos de uso	190
10.2. Secure Copy (<i>scp</i>)	190
10.2.1. El Protocolo <i>scp</i>	190
10.2.2. La aplicación <i>scp</i>	191
10.2.3. Ejemplos de uso	191
TEMA 8: EJECUTAR UN SISTEMA SEGURO	193
1. Una perspectiva sobre la seguridad del sistema	195
2. Algunos aspectos de seguridad a tomar en cuenta	197
2.1. Cierre de demonios de red no deseados.....	197
2.2. Evitar las amenazas de ingeniería social	198
2.3. Cumplimiento de las normas de seguridad	199
3. Las 10 cosas que nunca se deben hacer	199
No utilizar contraseñas simples o que se puedan adivinar con facilidad	200
No utilizar la cuenta de <i>root</i> a no ser que sea estrictamente necesario	200
No compartir las contraseñas	201
No creer ciegamente en los binarios proporcionados	201
No ignorar los archivos de registro	201
No dejar de actualizar el sistema durante mucho tiempo	202
No debemos olvidarnos de la seguridad física	202
No descuidar los permisos de los archivos.....	202
No olvidar la existencia de sitios web de seguridad	202
Nunca llevar a cabo una administración remota sin un <i>shell</i> seguro	203
4. Configuración del envoltorio TCP	204
4.1. Utilizar envoltorios TCP con <i>inetd</i>	204
4.2. Utilizar envoltorios TCP con <i>xinetd</i>	205
4.3. <i>/etc/hosts.allow</i> y <i>/etc/hosts.deny</i>	205
5. Cortafuegos: filtrado de paquetes IP	207
6. Otros elementos útiles para asegurar nuestro sistema	208
6.1. El servicio <i>finger</i> para descubrimiento de usuarios.....	208

6.2. Fortaleza de contraseñas con <i>crack</i>	208
6.3. Comprobación proactiva de contraseñas	209
6.3.1. Comprobador proactivo de contraseñas: <i>passwd+</i>	209
6.3.2. Comprobador proactivo de contraseñas: <i>anlpasswd</i>	210
6.3.3. Comprobador proactivo de contraseñas: <i>npasswd</i>	210
6.4. Mapeo de puertos con <i>nmap</i>	211
6.5. Sistemas de detección de intrusos	211
6.5.1. Detección de ataques en red con <i>snort</i>	212
6.6. Advertencias ante paquetes con <i>bugs</i>	212
6.7. Auditando vulnerabilidades con <i>nessus</i>	213
TEMA 9: COPIAS DE SEGURIDAD Y RECUPERACIÓN	215
1. Introducción a las copias de seguridad	216
1.1. Modelos de almacén de datos	216
Desestructurado	217
Completa + incremental.....	217
Espejo + diferencial	217
Protección continua de datos	217
1.2. Medios de almacenamiento	217
Cinta magnética	217
Disco duro	218
Disco óptico.....	218
Disquetes	218
Dispositivos de memoria no volátil	218
Servicios remotos de copia de seguridad	218
1.3. Administrar un almacén de datos	218
En línea.....	218
Cerca de línea.....	219
Fuera de línea	219
Cámara fuera del lugar.....	219
Centro de recuperación de datos	219
2. Concejos a tener en cuenta al realizar copias de seguridad	219
3. Planificación de las copias de seguridad	221
4. Copias de seguridad completas	222
5. Copias de seguridad incrementales	223
6. Aplicando compresión a las copias de seguridad	224
6.1 Problemas que presenta comprimir las copias de seguridad.....	225
7. Otras herramientas para realizar copias de seguridad en Linux.....	226
7.1. Copias de seguridad utilizando <i>dd</i>	226
7.2. Copias de seguridad utilizando <i>dump</i>	227
7.3. Copias de seguridad utilizando <i>cpio</i>	228
7.4. Copias de seguridad utilizando <i>afio</i>	229
7.5. Copias de seguridad utilizando <i>rsync</i>	229
7.6. Copias de seguridad utilizando <i>amanda</i>	230
8. ¿Qué hacer en caso de emergencia?	231
9. Revisar y recuperar sistemas de archivos	232
10. Recuperación del súper bloque	233
11. <i>MBR</i> dañado, infectado o corrupto	234
X. PRÁCTICAS DE LABORATORIO	237

Práctica 0: Programación de <i>shell scripts</i>	239
Práctica 1: Lenguaje <i>awk</i>	263
Práctica 2: Instalación del sistema operativo Linux	285
Práctica 3: Instalación de software	289
Práctica 4: Arranque y apagado del sistema	301
Práctica 5: Administración de usuarios y grupos	311
Práctica 6: Administración del sistema de archivos	323
Práctica 7: Administración de la red	337
Práctica 8: Copias de seguridad	349
XI. BIBLIOGRAFÍA	359

ÍNDICE DE FIGURAS

1.1. Interacción entre un núcleo (<i>kernel</i>), el software restante y el hardware.....	38
1.2. Esquema de núcleo <i>monolítico</i>	39
1.3. Esquema del funcionamiento de un <i>micronúcleo</i>	39
1.4. Esquema de interacción entre un <i>exonúcleo</i> y el software a través de las bibliotecas.....	40
1.5. Dennis Ritchie (derecha) y Ken Thompson (izquierda).....	41
1.6. pdp 7 de <i>DEC</i>	41
1.7. pdp 11 de <i>DEC</i>	42
1.8. <i>VAX</i> de <i>DEC</i>	42
1.9. Linus Torvalds.....	43
1.10. Richard Stallman.....	50
1.11. Ciclo de vida del sistema.....	56
2.1. Estructura del <i>MBR</i>	67
3.1. Esquema de elementos basados en texto de la familia de programas <i>apt</i>	89
3.2. Esquema básico utilizado para montar la mayoría de programas.....	98
5.1. Propiedad y grupo de un archivo.....	127
5.2. Esquema típico de creación de un usuario.....	130
5.3. Arquitectura de <i>Linux-PAM</i>	136
6.1. Utilizar <i>inodos</i> para asignar la información de los datos para los datos dentro de los grupos de bloque.....	141
6.2. Dispositivo con un sistema de archivos <i>ext2</i>	142
6.3. Esquema del árbol típico de directorios.....	145
6.4. Esquema de un enlace simbólico.....	152
6.5. Mostrar la propiedad y los permisos.....	157
6.6. Andrew Tridgell.....	162
7.1. Ubicación de TCP en la pila de protocolos.....	171
7.2. Ubicación de IP en la pila de protocolos.....	171
7.3. Ubicación de UDP en la pila de protocolos.....	171
7.4. Implementación de niveles de redes.....	172
7.5. Direccionamiento para la clase A.....	177
7.6. Direccionamiento para la clase B.....	177
7.7. Direccionamiento para la clase C.....	178
7.8. Direccionamiento para la clase D.....	178
7.9. Direccionamiento para la clase E.....	178
7.10. Esquema de ejemplo de red.....	182
7.11. Estructura de la capa de transporte <i>ssh</i>	188
9.1. Estructura de un servidor de copias de seguridad con <i>rsync</i>	229
9.2. Copia de seguridad de un archivo utilizando <i>rsync</i>	230

ÍNDICE DE TABLAS

1.1. Lista de condiciones que debe cumplir un programa para ser considerado <i>Open Source</i>	52
1.2. Los seis tipos de licencias generadas con combinaciones de las cuatro condiciones.....	55
6.1. Tipos de archivos de Linux según la naturaleza de su contenido.....	154
6.2. Atributos existentes en el sistema de archivos de Linux.....	154
6.3. Valores comúnmente usados para el <i>umask</i>	159
7.1. Máscaras de red estándar.....	178
8.1. Herramientas alternativas al comando <i>crack</i>	209
9.1. Opciones básicas del comando <i>cpio</i>	228

I. INTRODUCCIÓN

El Programa de Cooperación con Nicaragua (PCN) es un Proyecto de gran impulso orientado a la formación de capacidades académicas por parte de la Universidad de Alcalá, cuyo objetivo es contribuir al desarrollo de Nicaragua a través del fortalecimiento académico de futuros docentes de la Universidad Nacional Autónoma de Nicaragua, UNAN-León. Un Proyecto alimentado por una visión compartida sobre el papel y la posibilidad de la universidad como agente de desarrollo del conocimiento, la tecnología y la cultura y una filosofía de trabajo planificado, organizado y de medio-largo plazo.

En el año 1994 se creó el convenio entre el Departamento de Automática de la Universidad de Alcalá de Henares (UAH) y el Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua (UNAN-León) con el objetivo de impulsar la excelencia en la enseñanza académica en la carrera de Ingeniería en Sistemas de Información (antes Licenciatura en Computación), lo cual ha contribuido y sigue contribuyendo con la formación de mejores profesionales en el área de la informática.

Como consecuencia del gran impacto que están teniendo las redes y las tecnologías de comunicaciones hoy en día, el departamento de Computación de la UNAN-León decidió agregar, con el apoyo del convenio que tiene con la Universidad de Alcalá de Henares, una nueva opción de carrera profesional: la carrera de Ingeniería en Telemática, la cual contempla un plan de estudios que requiere de un periodo de cinco años de estudios superiores (los tres primeros años son semejantes con la carrera de Ingeniería en sistemas de información) y una dedicación de tiempo completo por parte del estudiante.

En el presente documento se aborda el desarrollo del Plan Docente de la asignatura de *Administración de Sistemas Operativos* que se imparte en el segundo semestre del cuarto año de la carrera (en general octavo semestre de la carrera) de Ingeniería en Telemática de la UNAN-León.

Además de lo mencionado anteriormente también se presentan los siguientes aspectos:

- La situación actual de la asignatura.
- Su relación con otras asignaturas.
- La metodología empleada.
- El material didáctico de apoyo para impartirla.
- El sistema de evaluación.
- La planificación temporal.
- La presentación de los contenidos teóricos y su respectiva bibliografía.
- El desarrollo de las prácticas y su respectiva bibliografía.

El plan docente está compuesto por dos partes. En la primera parte se desarrollan los contenidos teóricos que son necesarios para cumplir con los objetivos de la asignatura y son esenciales para la comprensión de las prácticas. En la segunda parte se desarrolla la parte correspondiente a las prácticas de laboratorio que permitirán a los estudiantes reforzar y aplicar los conocimientos adquiridos en las clases teóricas.

II. OBJETIVOS

Los objetivos del presente proyecto docente son

Para el docente que impartirá la asignatura:

- Generar un instrumento documentado que le sirva de guía para impartir la asignatura de *Administración de Sistemas Operativos*, para asegurar de esta manera una mejor planificación y organización de la misma.
- Utilizar este documento como guía para proporcionar a los estudiantes los conocimientos teórico-prácticos para: Instalar, configurar y administrar un sistema operativo tipo Unix.

Para el estudiante:

- Adquirir los conocimientos teóricos relacionados con la *Administración de Sistemas Operativos* y poner en práctica dichos conocimientos a lo largo de toda la asignatura.
- Introducirlo en las tareas que debe realizar un administrador de un sistema operativo tipo Unix, para conseguir un aprovechamiento óptimo de los recursos de que dispone.
- Conocer las técnicas de administración de sistemas Unix y la provisión de algunos servicios soportados por esta plataforma.

III. SITUACIÓN ACTUAL DE LA ASIGNATURA EN EL PÉNSUL ACADÉMICO DE INGENIERÍA EN TELEMÁTICA

La asignatura de *Administración de Sistemas Operativos*, se impartirá en la carrera de Ingeniería en Telemática de la UNAN-León, la cual es ofrecida por el departamento de computación de la Facultad de Ciencias de dicha universidad.

La carrera se desarrolla a lo largo de diez semestres de estudios (cinco años académicos/lectivos). Esta asignatura se imparte en el octavo semestre del plan de estudios actual (segundo semestre del cuarto año de la carrera).

La asignatura consta de un total de sesenta y cuatro horas (tres créditos académicos), las cuales incluyen tanto tiempo destinado a la parte teórica como a la parte práctica, distribuidas a lo largo de dieciséis semanas lectivas.

División de horas destinadas para teoría y para práctica:

	Horas semanales	Horas Semestrales
Teoría	2 horas	2 horas semanales * 16 semanas = 32 horas
Práctica	2 horas	2 horas semanales * 16 semanas = 32 horas
64 horas totales por semestre		

Para lograr un total aprendizaje de la asignatura de *Administración de Sistemas Operativos* el estudiante deberá tener ciertos conocimientos previos adquiridos en la asignatura de previo requisito (*Sistemas operativos II*).

En la siguiente sección se citan cuales son las asignaturas que de modo directo se relacionan con la asignatura de *Administración de Sistemas Operativos*, sin embargo todo estudiante, debe haber desarrollado las siguientes competencias:

Parte teórica:

- Poseer los conocimientos básicos generales de teoría de sistemas operativos, tanto de su estructura interna como a nivel de usuario.

Parte práctica:

- Manejo de los comandos básicos del intérprete de órdenes de un sistema Unix.

IV. RELACIÓN CON OTRAS ASIGNATURAS

El proceso de aprendizaje de la asignatura de *Administración de Sistemas Operativos*, requiere de otros conocimientos que una vez que el estudiante llegue a esta asignatura deberá tener muy claros y afianzados para de esta forma lograr una mejor comprensión de la misma. Estos conocimientos son adquiridos por los estudiantes a lo largo de su formación académica en asignaturas antecedentes.

Por este motivo se hace importante plasmar la relación de la asignatura: *Administración de Sistemas Operativos* con otras asignaturas.



Arquitectura de computadores:

- Esta asignatura se imparte en el quinto semestre del plan de estudios actual (primer semestre del tercer año de la carrera). Proporciona el conocimiento de los conceptos relacionados con la arquitectura de un computador, haciendo énfasis en los componentes de la misma y basándose en la arquitectura propuesta por Von Newman.

Sistemas Operativos I y II:

- Estas asignaturas se imparten en el sexto y séptimo semestre del plan de estudios actual, respectivamente. Permitirá conocer la manera de trabajar de los sistemas operativos y sobre todo, formará las bases para comprender y aplicar los conceptos de núcleo del sistema, ejecución de procesos, intérprete de órdenes y estructura del sistema Unix; en un entorno local al sistema, conceptos importantes para la correcta asimilación de los nuevos términos de estudio que se utilizan en la asignatura de *Administración de Sistemas Operativos*.

Mediante la articulación de las asignaturas anteriores, que son antecedentes a la asignatura de *Administración de Sistemas Operativos* se podrá articular la asimilación, por parte del estudiante, de los conceptos teórico/práctico abordados en la *Administración de Sistemas Operativos*.

V. CONTENIDO DEL TEMARIO

TEMA 0: PRESENTACIÓN DE LA ASIGNATURA

- Presentación del docente
- Información de la asignatura
- Objetivos de la asignatura
- Temario
- Material de estudio
- Metodología de evaluación
- Tutorías

TEMA 1: INTRODUCCIÓN

- Repaso de conceptos importantes sobre sistemas operativos
- Introducción a Unix
- Software libre y Linux
- Introducción a la administración

TEMA 2: INSTALACIÓN BÁSICA

- Conceptos necesarios previos a la instalación
- Tareas de preparación para la instalación de Linux
- Procedimientos posteriores a la instalación

TEMA 3: INSTALACIÓN, ACTUALIZACIÓN Y COMPILACIÓN DE PROGRAMAS

- Actualización de software.
- Procedimiento general de actualización
- Sistema de gestión de paquetes *RPM*
- Sistemas de gestión de paquetes *.deb*
- Software no proporcionado en paquetes

TEMA 4: INICIO Y CIERRE DEL SISTEMA

- Inicio del sistema
- Inicio e inicialización del sistema
- Modo de un solo usuario
- Cierre del sistema

TEMA 5: ADMINISTRACIÓN DE USUARIOS, GRUPOS Y PERMISOS

- Tipos de cuentas de usuario
- Usuarios y *uids*
- Pseudo-conexiones
- Gestión de usuarios y grupos
- Operaciones con las cuentas de usuario
- Otro método de autenticación: *PAM*

TEMA 6: EL SISTEMA DE ARCHIVOS

- Concepto de archivo y de sistema de archivos
- Los *inodos*
- El *superbloque*
- El sistema de archivos *ext2*
- El sistema de archivos *ext3*
- El estándar de jerarquía del sistema de archivos
- Algunos directorios interesantes
- Nombres de archivos y directorios
- Tipos de archivos
- Atributos existentes en el sistema de archivos de Linux
- Propiedad y permisos de los archivos
- Puntos adicionales del sistema de archivos
- Sistemas de archivos distribuidos (*DFS*)

TEMA 7: SISTEMAS DE RED

- Introducción a TCP/IP
- Servicios sobre TCP/IP
- ¿Qué es TCP/IP?
- La pila de protocolos IP
- Dispositivos físicos (hardware) de red
- Conceptos TCP/IP
- Direcciones TCP/IP
- Componentes de la dirección de red
- Configurar la red
- Aplicaciones seguras

TEMA 8: EJECUTAR UN SISTEMA SEGURO

- Una perspectiva sobre la seguridad del sistema
- Algunos aspectos de seguridad a tomar en cuenta
- Las 10 cosas que nunca se deben hacer
- Configuración del envoltorio TCP
- Cortafuegos: filtrado de paquetes IP
- Otros elementos útiles para asegurar nuestro sistema

TEMA 9: COPIAS DE SEGURIDAD Y RECUPERACIÓN

- Introducción a las copias de seguridad
- Concejos a tener en cuenta al realizar copias de seguridad
- Planificación de las copias de seguridad
- Copias de seguridad completas
- Copias de seguridad incrementales
- Aplicando compresión a las copias de seguridad
- Otras herramientas para realizar copias de seguridad en Linux
- ¿Qué hacer en caso de emergencia?
- Revisar y recuperar sistemas de archivos
- Recuperación del súper bloque
- *MBR* dañado, infectado o corrupto

VI. METODOLOGÍA DIDÁCTICA Y MATERIAL DIDÁCTICO A UTILIZAR

Metodología didáctica

➤ Parte Teórica:

La metodología que se utilizará para impartir la asignatura de *Administración de Sistemas Operativos* serán las lecciones magistrales con una duración de dos horas, una vez a la semana durante las cuales se abordará de forma planificada el contenido teórico desarrollado en este plan. Cada sesión de dos horas se planificará en dos bloques de sesenta minutos cada uno.

Las lecciones serán complementadas con cualquier técnica de enseñanza como pueden ser diagramas, ejemplos, tablas, etc. que ayude a una mejor comprensión de los conceptos teóricos expuestos. Quedará a libre elección del docente cualquier criterio que ayude a incentivar el interés de los estudiantes hacia la asignatura ya sea mediante trabajos en grupo, presentaciones individuales, trabajos investigativos, etc.

➤ Parte Práctica:

En las prácticas de laboratorio se realizará una explicación de la misma al iniciar cada una de las sesiones nuevas de laboratorio, y puesto que las prácticas están guiadas el docente sólo deberá aclarar dudas o resolverá algún otro problema específico. Quedará a libre elección del docente cualquier criterio que ayude a incentivar el interés de los estudiantes para mejorar la calidad de las prácticas que estos entregan ya sea a través de mejoras en las mismas, entrega de versiones con añadidos, etc.

Material Didáctico:

Tanto en la parte teórica como en la práctica se utilizará el mismo material didáctico el cual será:

- La pizarra para las explicaciones y ejemplos que sea necesario.
- Presentaciones teóricas/prácticas con diapositivas utilizando para tal fin los medios didácticos necesarios como son: retroproyector y láminas de acetato, o laptops y data show.
- La asignatura contará con una página web en la cual se colocará todo el material didáctico correspondiente tanto para la parte teórica como para la parte práctica, así como también material electrónico de apoyo y enlaces a diversos sitios de interés.
- Acceso a la bibliografía básica para este tema y a la información impresa del material en la web la cual estará disponible en la biblioteca del departamento.

VII. PLANIFICACIÓN TEMPORAL

La planificación de la asignatura, para que sea de una manera objetiva, debe de realizarse año con año en función del calendario académico del que se disponga para poder, a partir de este, calcular el número de sesiones lectivas de las cuales se dispone.

La asignatura de *Administración de Sistemas Operativos* se imparte en el segundo semestre del cuarto año de la carrera y está constituida de tres créditos académicos. Normalmente el segundo semestre consta con un total de dieciséis semanas hábiles, pero considerando los días festivos y dando margen a cualquier otra eventualidad que pudiese suceder como son: exámenes retrasados, protestas nacionales, etc. podemos deducir que se tienen catorce semanas netas para cumplir con el contenido de la asignatura.

Tanto la parte teórica como la práctica, constan de una sesión de dos horas a la semana, por catorce semanas que hemos asumido como netas, suman un total de veintiocho horas para la teoría y veintiocho horas para las prácticas.

El número de horas asignadas a cada tema y a cada práctica, se ha calculado en base a la profundidad con que se quiere abordar cada tema. De tal manera que la planificación temporal tanto para la teoría como para las prácticas es la siguiente:

Planificación temporal de la parte teórica:

N° de tema	Tema	Horas
0	Presentación de la asignatura	1
1	Introducción	3
2	Instalación básica	2
3	Instalación, actualización y compilación de programas	2
4	Inicio y cierre del sistema	2
5	Administración de usuarios, grupos y permisos	4
6	El sistema de archivos	6
7	Sistemas de red	4
8	Ejecutar un sistema seguro	2
9	Copias de seguridad y recuperación	2
Horas totales de teoría:		28

Planificación temporal de la parte práctica:

N° de práctica	Práctica	Horas
0	Programación de <i>shell scripts</i>	4
1	Lenguaje AWK	4
2	Instalación del sistema operativo Linux	4
3	Instalación de software	2
4	Arranque y apagado del sistema	2
5	Administración de usuarios y grupos	4
6	Administración del sistema de archivos	4
7	Administración de la red	2
8	Copias de seguridad	2
Horas totales de prácticas:		28

VIII. METODOLOGÍA DE EVALUACIÓN

La asignatura está compuesta por las clases teóricas y las prácticas que son realizadas en el laboratorio. A lo largo del semestre se realizarán dos evaluaciones parciales (primero y segundo parcial) que corresponderán al 60% de la nota final y el restante 40% se obtendrá en una evaluación al final del semestre. Cada una de las evaluaciones parciales se desglosa de la siguiente manera:

Evaluación del primer parcial:

➤ Examen teórico.....	70%
➤ Evaluación práctica.....	+30%
➤ Total.....	100%

Evaluación del segundo parcial:

➤ Examen teórico.....	70%
➤ Evaluación práctica.....	+30%
➤ Total.....	100%

Las notas del primero y segundo parcial se suman, se dividen entre dos y se multiplican por 0.6 (ya que equivalen al 60% de la nota final) para de esta forma obtener la nota de entrada al examen final. El examen final tiene un valor de cien puntos que luego se multiplican por 0.4 (para obtener el 40% al que equivale) y el resultado se suma con la nota de entrada.

Para aprobar la asignatura el alumno debe obtener una nota mayor o igual a sesenta. Aquellos estudiantes cuya suma del primero y segundo parcial dividido entre dos sea inferior a cincuenta no tendrán derecho a hacer el examen final. Queda a criterio del docente evaluar a alumnos que cumplan la condición antes mencionada en los casos en el que él considere conveniente.

Evaluación final/semestral:

➤ Nota acumulada	$((1er\ parcial + 2do\ parcial) \div 2) * 0.6$	≥ 50
➤ Examen final/semestral	$+ (nota\ del\ examen\ final/semestral) * 0.4$	
➤ Total	$\frac{\hspace{10em}}{nota\ final}$	≥ 60

Evaluación de la teoría:

El examen teórico que se realiza en cada parcial correspondiente al 70% constará de preguntas/cuestiones teóricas sobre todos los temas vistos en clase que permitirán evaluar el grado de comprensión de los conceptos básicos que el estudiante ha adquirido durante el transcurso de la asignatura.

Evaluación de las prácticas:

El 30% correspondiente a las evaluaciones prácticas se obtendrá a través de la entrega en tiempo y forma de las soluciones de las guías de laboratorio; y la defensa ante el docente de las mismas, todo esto debe hacerse en los laboratorios asignados para tal fin. Queda a libre elección del docente la realización de un examen práctico al final de cada parcial. Todas las entregas y defensas de prácticas al docente deberán ser evaluadas. Aquellos estudiantes que no entreguen las prácticas en tiempo y forma deberán ser sancionados según el criterio que decida tomar el docente.

El 40% correspondiente a la evaluación final/semestral se obtendrá a través de un examen en el cual se contemplarán preguntas/cuestiones teóricas sobre todos los temas abordados en clase a lo largo del semestre y ejercicios/cuestiones prácticos abordados en las diferentes sesiones de laboratorio.

IX. DESARROLLO DEL TEMARIO

TEMA 0: PRESENTACIÓN DE LA ASIGNATURA

Presentación del docente

- El docente deberá presentarse formalmente ante los estudiantes.
- También deberá bríndales toda la información de contacto como puede ser: correo electrónico, teléfono de oficina, pagina web personal, etc., necesaria para que ellos puedan comunicarse con él.

Información de la asignatura

- El docente deberá mencionar los horarios de clases correspondientes tanto para la parte teórica como para la parte practica.
- Se darán a conocer las ubicaciones de las aulas en las cuales se impartirá la teoría y el laboratorio en el cual se impartirá la práctica.
- En el caso de que exista algún cambio importante como: cambio de horarios, cambio de laboratorio, etc. este deberá ser informado a los estudiantes en este punto.

Objetivos de la Asignatura

- Se mencionarán los objetivos que se pretenden alcanzar al cursar la asignatura.
- Se presentarán los logros que serán alcanzados al finalizar la asignatura

Temario

- Se hará una breve introducción/descripción de cada uno de los temas que van a ser desarrollados a lo largo de la asignatura.

Referencia: ver temas abordados en la asignatura en página número 25.

- Se indicará la planificación temporal para cada uno de los temas teóricos, así como también la planificación temporal para el desarrollo de las prácticas de laboratorio.

Referencia: ver planificación temporal de la asignatura en página número 28.

- Se hará una breve introducción/descripción de cada una de las prácticas de laboratorio que serán realizadas a lo largo de la asignatura.

Material de estudio

- Se informará al alumno del material (transparencias) que se utilizará como guía para el desarrollo de la asignatura.
- Se dará a conocer la dirección de la página Web de la asignatura en la cual se colocará toda la información necesaria tanto para la parte teórica como para la parte práctica así como cualquier otro elemento(s) que el docente considere necesario.
- Se presentará la bibliografía básica y complementaria utilizada para el desarrollo de la asignatura. En general para la parte teórica el libro básico será: *Matthias Kalle Dalheimer y Matt Welsh, "Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada", Editorial Anaya Multimedia, 2006.* Y el libro complementario será: *Dee-Ann LeBlanc, "Administración de sistemas LINUX La biblia". Editorial*

ANAYA MULTIMEDIA, 2001. Para la parte práctica se usará como libro básico: Sebastián Sánchez Prieto, Óscar García Población, “UNIX y LINUX Guía Práctica Tercera edición”. Editorial Ra-Ma, 2005. Y como libro complementario: Iñaki Alegría Loinaz, Roberto Cortiñas Rodríguez, Aitzol Ezeiza Ramos, “Linux Administración del sistema y la red”, Editorial Prentice Hall, 2005.

- De existir otras fuentes de información (fuentes de interés, enlaces de apoyo, recursos, etc.) distintas a las mencionadas en los apartados anteriores estas también deberán ser presentadas en este punto.

Metodología de evaluación

- Se explicará al estudiante la metodología de evaluación empleada durante el desarrollo de la asignatura, así como también todos aquellos requisitos que deben cumplir para aprobar la asignatura.

Referencia: ver la metodología de evaluación a emplear en la asignatura en la página número 29.

Tutorías

- Todos los estudiantes podrán hacer consultas al docente fuera del aula de clases en el horario que dicho docente disponga para consultas.
- Por lo tanto se deberá brindar a los estudiantes el horario para estas consultas.
- Las tutorías sólo se deberán llevar a cabo en horarios de oficina. En casos de tutorías diferentes a la antes mencionada entonces el estudiante deberá hacer uso de la información de contacto provista por el docente en la parte correspondiente a la presentación del mismo.

TEMA 1: INTRODUCCIÓN

Objetivos

- Repasar de manera breve los aspectos básicos sobre sistemas operativos.
- Promover conocimientos sobre la historia de Unix y Linux.
- Estudiar el concepto de distribución y conocer algunas de ellas.
- Estudiar el concepto de software libre y lo que éste representa.
- Presentar las tareas, labores y responsabilidades del administrador de sistemas.

Contenido

1. Repaso de conceptos importantes sobre sistemas operativos
 - 1.1. ¿Qué es un sistema operativo?
 - 1.2. Concepto de proceso
 - 1.3. Concepto de núcleo
 - 1.3.1. Tipos de núcleo
 - 1.4. Conceptos de: Multitarea, multiusuario y multiplataforma
2. Introducción a Unix
 - 2.1. Historia de Unix
 - 2.2. Aparición de Linux
 - 2.3. Distribuciones de Linux
 - 2.3.1. Principales distribuciones
3. Software libre y Linux
 - 3.1. *GNU*
 - 3.2. Libertad y coste
 - 3.3. *Open Source*
 - 3.4. Licencias en el software libre
 - 3.4.1. Licencias tipo *BSD*
 - 3.4.2. La licencia pública general de *GNU (GNU GPL)*
 - 3.5. Licencias de otros recursos libres
 - 3.5.1. Licencia de documentación libre de *GNU*
 - 3.5.2. Licencias de *Creative Commons*
4. Introducción a la administración
 - 4.1. Ciclo de vida del sistema
 - 4.2. El administrador del sistema
 - 4.3. La primera regla del administrador
 - 4.4. Responsabilidades del administrador
 - 4.5. Seguridad en la administración

Bibliografía

Básica

- Sebastián Sánchez Prieto, Óscar García Población, “UNIX y LINUX Guía Práctica Tercera edición”. Editorial Ra-Ma, 2005.
- Jordi Mas i Hernández, “Software Libre: técnicamente viable, económicamente sostenible y socialmente justo Primera edición”, infonomia Red de innovadores, 2005.
<http://www.softcatala.org/~jmas/swl/lilibrejmas.pdf>
- M Carling, Stephen Degler, James Dennis, “Administración de Sistemas Linux Guía Avanzada”. Editorial Prentice Hall, 2000.

- Sebastián Sánchez Prieto, “Sistemas operativos, textos universitarios, segunda edición”, Editorial universidad de Alcalá.

Complementaria

- Sistema operativo.
http://es.wikipedia.org/wiki/Sistema_operativo
- Proceso (informática).
http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29
- Núcleo (informática).
[http://es.wikipedia.org/wiki/N%C3%BAcleo_\(inform%C3%A1tica\)](http://es.wikipedia.org/wiki/N%C3%BAcleo_(inform%C3%A1tica))
- Núcleo monolítico.
http://es.wikipedia.org/wiki/N%C3%BAcleo_monol%C3%ADtico
- Micronúcleo
<http://es.wikipedia.org/wiki/Micron%C3%BAcleo>
- Núcleo híbrido.
http://es.wikipedia.org/wiki/N%C3%BAcleo_h%C3%ADbrido
- Exonúcleo.
<http://es.wikipedia.org/wiki/Exon%C3%BAcleo>
- Distribución Linux.
http://es.wikipedia.org/wiki/Distribuci%C3%B3n_Linux
- Red Hat.
http://es.wikipedia.org/wiki/Red_Hat
<http://www.redhat.es/>
- Fedora (distribución Linux).
http://es.wikipedia.org/wiki/Fedora_Core
<http://fedoraproject.org/>
- SUSE Linux.
http://es.wikipedia.org/wiki/SUSE_Linux
http://es.opensuse.org/Bienvenidos_a_openSUSE.org
- Mandriva Linux.
http://es.wikipedia.org/wiki/Mandriva_Linux
<http://www.mandriva.com/>
- Debian.
<http://es.wikipedia.org/wiki/Debian>
<http://www.debian.org/index.es.html>
- GnuLinEx.
<http://es.wikipedia.org/wiki/GnuLinEx>
<http://www.linex.org/joomlaex/>
- Ubuntu (distribución Linux).
[http://es.wikipedia.org/wiki/Ubuntu_\(distribuci%C3%B3n_Linux\)](http://es.wikipedia.org/wiki/Ubuntu_(distribuci%C3%B3n_Linux))
<http://www.ubuntu.com/>
- Creative Commons.
http://es.wikipedia.org/wiki/Creative_Commons
<http://es.creativecommons.org/licencia/>
- El sistema operativo GNU.
<http://www.gnu.org/home.es.html>

1. Repaso de conceptos importantes sobre sistemas operativos

Antes de introducirnos de lleno en la administración de sistemas operativos es necesario recordar algunos conceptos que serán de gran utilidad para la comprensión de cada uno de los elementos que nos permitirán la correcta administración de un sistema operativo Linux. En la siguiente sección se describen los conceptos considerados como más relevantes antes de empezar a hablar de la administración de sistemas.

1.1. ¿Qué es un sistema operativo?

Un sistema operativo es un software de sistema, es decir, un conjunto de programas de computadora destinado a permitir una administración eficaz de sus recursos. Comienza a trabajar cuando se enciende el computador, y gestiona el hardware de la máquina desde los niveles más básicos, permitiendo también la interacción con el usuario.

Los sistemas operativos, en su condición de capa software, posibilitan y simplifican el manejo de la computadora, desempeñan una serie de funciones básicas esenciales para la gestión del equipo. Entre las más destacables, cada una ejercida por un componente interno, podemos mencionar las siguientes:

- Proporcionar comodidad en el uso de un computador.
- Gestionar de manera eficiente los recursos del equipo, ejecutando servicios para los procesos o programas.
- Brindar una interfaz al usuario, ejecutando instrucciones u órdenes.
- Permitir que los cambios, debido al desarrollo del propio sistema operativo, se puedan realizar sin interferir con los servicios que ya se prestaban (conocido como evolutividad).

Un sistema operativo desempeña cinco funciones básicas en la operación de un sistema informático como son: Suministro de interfaz al usuario, administración de recursos, administración de archivos, administración de tareas y servicio de soporte y utilidades.

1.2. Concepto de proceso

Un proceso es un programa en ejecución. Se trata de una entidad dinámica, a diferencia del programa que es una entidad estática.

Un proceso se genera cuando el sistema operativo carga un programa de disco en memoria principal y le asigna los recursos que necesita para ejecutarse. Así mismo, el sistema operativo crea una estructura u objeto para gestionar la información de cada uno de los trabajos existentes en el sistema.

Los procesos son creados y destruidos por el sistema operativo, así como también éste se debe hacer cargo de la comunicación entre procesos, pero lo hace a petición de otros procesos. El mecanismo por el cual un proceso crea otro proceso se denomina bifurcación (*fork*).

1.3. Concepto de núcleo

El núcleo, también conocido como *kernel* (ver figura 1.1), es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora, o en forma más básica, es el encargado de gestionar recursos a través de servicios de llamadas al sistema.

Como hay muchos programas y el acceso al hardware es limitado, el núcleo también se encarga de decidir qué programa podrá hacer uso de un dispositivo de hardware y durante cuánto tiempo.

Acceder al hardware directamente puede ser realmente complejo, por lo que los núcleos suelen implementar una serie de abstracciones del hardware. Esto permite ocultar la complejidad, y proporciona una interfaz limpia y uniforme al hardware subyacente, lo que facilita su uso para el programador.

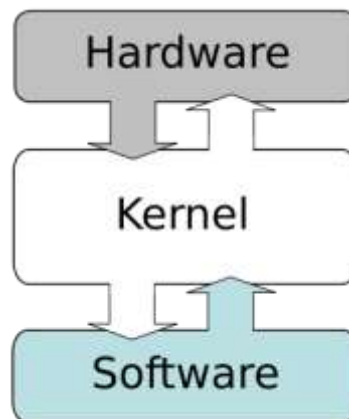


Figura 1.1: Interacción entre un núcleo (*kernel*), el software restante y el hardware

No necesariamente se necesita un núcleo para usar una computadora. Los programas pueden cargarse y ejecutarse directamente en una computadora *vacía*, siempre que sus autores quieran desarrollarlos sin usar ninguna abstracción del hardware ni ninguna ayuda del sistema operativo. Ésta era la forma normal de usar muchas de las primeras computadoras: Para usar distintos programas se tenía que reiniciar y reconfigurar la computadora cada vez. Con el tiempo, se empezó a dejar en memoria pequeños programas auxiliares, como el cargador y el depurador, o se cargaban desde memoria de sólo lectura. A medida que se fueron desarrollando, se convirtieron en los fundamentos de lo que llegarían a ser los primeros núcleos de sistema operativo.

1.3.1. Tipos de núcleo

Principalmente existen cuatro grandes tipos de núcleos: Los núcleos *monolíticos*, los *micronúcleos*, los núcleos *híbridos* y los *exonúcleos*. En la siguiente sección se detalla cada uno de ellos.

Los núcleos *monolíticos* (ver figura 1.2):

Los sistemas operativos de núcleo *monolítico* tienen la característica de poseer un núcleo grande y complejo, que engloba todos los servicios del sistema. El sistema operativo se ejecuta en modo supervisor, con las interrupciones deshabilitadas y en general tiene un rendimiento mucho mayor que un *micronúcleo*. Todo cambio a realizar en cualquier servicio implementado en el núcleo requiere la recompilación del mismo y el reinicio del sistema operativo para que los nuevos cambios sean aplicados y tengan efecto. Como ejemplos de sistemas operativos de núcleo *monolítico* podemos citar a Unix y Linux.

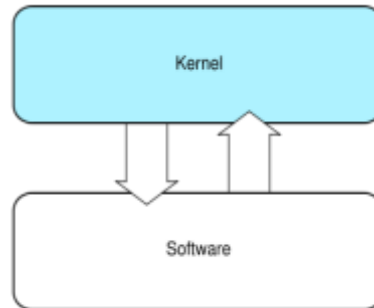


Figura 1.2: Esquema de núcleo *monolítico*

Los *Micronúcleos* o *Microkernel* (ver figura 1.3):

Es un tipo de núcleo de un sistema operativo que provee un conjunto de primitivas o llamadas al sistema mínimas para implementar servicios básicos como: Espacios de direcciones, comunicación entre procesos y conmutación de la CPU entre procesos; todos los otros servicios, como pueden ser: La gestión de la memoria, el sistema de archivos, la gestión de dispositivos de E/S, etc., que en general son provistos por el núcleo, se ejecutan como procesos servidores en espacio de usuario. Las principales ventajas de utilizar un sistema operativo que implemente un *micronúcleo* son: La reducción de la complejidad, la descentralización de los fallos (un fallo en una parte del sistema no lo colapsaría por completo) y la facilitación para crear y depurar controladores de dispositivos. Dentro de las principales dificultades que podemos citar están: La complejidad en la sincronización de todos los módulos que componen el *micronúcleo* y su acceso a la memoria, así como también la integración con las aplicaciones. Como ejemplos de sistemas operativos que implementan *micronúcleos* encontramos a Minix y Hurd.

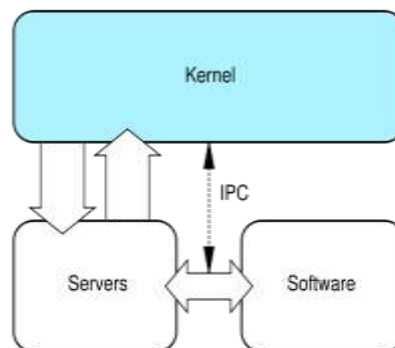


Figura 1.3: Esquema del funcionamiento de un *micronúcleo*

Los núcleos **híbridos**:

Fundamentalmente son *micronúcleos* que tienen algo de código *no esencial*, en espacio de núcleo para que éste se ejecute más rápido de lo que lo haría si estuviera en espacio de usuario. La mayoría de sistemas operativos modernos pertenecen a esta categoría, siendo los más populares aquellos pertenecientes a la familia de sistemas operativos de Microsoft Windows.

Los *Exonúcleos* o *exokernel* (ver figura 1.4):

Los *exonúcleos*, también conocidos como sistemas operativos verticalmente estructurados, representan una aproximación radicalmente nueva al diseño de sistemas operativos. Se tratan de núcleos que son extremadamente pequeños, ya que limitan expresamente su funcionalidad a la protección y compartición de los recursos. Toda su funcionalidad deja de estar residente en memoria y pasa a estar afuera de ella ubicándose en librerías dinámicas. La finalidad de un

exonúcleo es permitir a una aplicación que solicite una región específica de la memoria, un bloque de disco concreto, etc., y simplemente asegurarse que los recursos pedidos están disponibles, y que la aplicación tiene derecho a acceder a ellos. Debido a que el *exonúcleo* sólo proporciona una interfaz al hardware de muy bajo nivel, careciendo de todas las funcionalidades de alto nivel de otros sistemas operativos, éste es complementado por una *biblioteca de sistema operativo*.

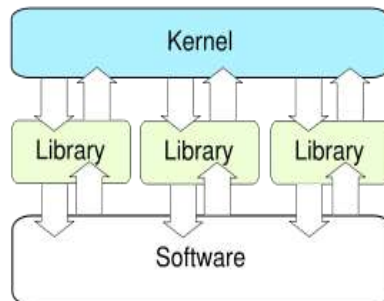


Figura 1.4: Esquema de interacción entre un *exonúcleo* y el software a través de las bibliotecas

1.4. Conceptos de: Multitarea, multiusuario y multiplataforma

Las ventajas de utilizar el sistema operativo Linux, se derivan precisamente de su potencia y flexibilidad. Estas dos propiedades son el resultado de las muchas funciones incorporadas al sistema y que permiten utilizarlo en el momento mismo en que se ejecuta. Los siguientes apartados repasan detenidamente algunas de estas características.

Multitarea

Es una característica de los sistemas operativos modernos que se refiere a la posibilidad de ejecutar varios programas a la vez, compartiendo uno o más procesadores, sin los inconvenientes de tener que detener la ejecución de cada aplicación. La mayoría de variantes de Unix integran un tipo de multitareas llamado *multitarea preferente*, es decir, que cada programa tiene garantizada la oportunidad de ejecutarse y se ejecuta hasta que el sistema operativo da prioridad a la ejecución de otro programa. Esta es precisamente el tipo de multitarea que incorpora Linux.

Multiusuario

Se refiere a la posibilidad de que varios usuarios, cada uno con ciertos niveles de permisos, accedan a las aplicaciones o recursos del sistema desde un único PC. La capacidad de Linux para asignar el tiempo de microprocesador simultáneamente a varias aplicaciones, permite ofrecer acceso a varios usuarios a la vez, ejecutando cada uno de ellos una o varias aplicaciones. La gran ventaja que les aporta a los sistemas Linux el poseer características como multitarea y multiusuario es que más de un usuario puede trabajar con la misma versión de la aplicación al mismo tiempo y desde el mismo terminal o desde terminales distintos. Sin embargo no debemos confundir esta capacidad con el hecho de que varios usuarios puedan actualizar el mismo archivo simultáneamente, algo que podría llevar a la confusión y al caos total y por ello resulta indeseable.

Multiplataforma

Se refiere a la capacidad que tiene un sistema operativo de ejecutarse en diferentes arquitecturas hardware.

2. Introducción a Unix

En esta sección abarcaremos un poco de la historia de los sistemas Unix, la cual es necesaria para poder describir como se produce la aparición de Linux, ya que este último posee orígenes compartidos con Unix. Terminaremos explicando el concepto de distribución Linux y además describiremos cada una de las principales distribuciones de Linux que existen actualmente.

2.1. Historia de Unix

Los antecedentes de Unix se remontan a 1964. En este año, *Bell Telephone Laboratories de AT&T*, *General Electric Company* y el *MIT* (Instituto Tecnológico de Massachusetts) se plantearon desarrollar un nuevo sistema operativo en tiempo compartido para una máquina *GE 645* (de *General Electric*) al que denominaron *MULTICS*. Los objetivos buscados inicialmente consistían en proporcionar a un conjunto amplio de usuarios una capacidad de computación grande y la posibilidad de almacenar y compartir grandes cantidades de datos si éstos lo deseaban. Todos esos objetivos eran demasiado ambiciosos para la época, sobre todo por las limitaciones del hardware. Como consecuencia de ello, los trabajos en el nuevo sistema operativo iban muy retrasados. Debido a eso, *Bell Laboratories* decidió dar por terminada su participación en el proyecto. A pesar del fracaso de *MULTICS*, las ideas empleadas para su diseño no cayeron en el olvido, sino que influyeron mucho en el desarrollo de Unix y de otros sistemas operativos posteriores.

Ken Thompson (ver figura 1.5), uno de los miembros del *Computing Science Research Center* de los *Laboratorios Bell*, encontró un computador *DEC (Digital Equipment Corporation) PDP-7* (ver figura 1.6) inactivo y se puso a desarrollar en él un juego denominado *Space Travel*. El desarrollo de ese juego propició que Thompson adquiriese muchos conocimientos relacionados con la máquina en la que estaba trabajando. Con objeto de crear un entorno de trabajo agradable, Thompson, al que posteriormente se le unió Dennis Ritchie (ver figura 1.5), se propuso la creación de un nuevo sistema operativo, al que denominó Unix. Ritchie había trabajado anteriormente en el proyecto *MULTICS*, de mucha influencia en el nuevo sistema operativo. Como ejemplos de esa influencia podemos citar la organización básica del sistema de archivos, la idea del intérprete de órdenes (*shell*) como proceso de usuario (en sistemas anteriores, el intérprete de órdenes formaba parte del propio núcleo del sistema operativo), e incluso el propio nombre Unix deriva de *MULTICS*.

MULTICS: MULTiplexed **I**nformation and **C**omputing **S**ervice.

UNICS: UNiplexed **I**nformation and **C**omputing **S**ervice.



Figura 1.5: Dennis Ritchie (derecha) y Ken Thompson (izquierda)



Figura 1.6: pdp 7 de DEC

Realmente, el término *UNICS* se empleó por la similitud de esta palabra con la palabra inglesa *eunuc*, con lo cual se venía a indicar que este nuevo sistema operativo era un *MULTICS* castrado. Posteriormente, *UNICS* dio lugar al nombre definitivo Unix. El nuevo sistema también se vio influenciado por otros sistemas operativos, tales como el *CTSS (Compatible Time Sharing System)* del *MIT* y el sistema *XDS-940 (Xerox Data System)* de la universidad de California en Berkeley.

Aunque esta primera versión de Unix prometía mucho, su potencial no pudo demostrarse hasta que se utilizó en un proyecto real. Así pues, mientras se planeaban las pruebas para patentar el nuevo producto, éste fue trasladado a un computador *PDP-11* (ver figura 1.7) de *Digital* en una segunda versión. En 1973 el sistema operativo fue reescrito en lenguaje C en su mayor parte. C es un lenguaje de alto nivel (las versiones anteriores del sistema operativo habían sido escritas en lenguaje ensamblador), lo que propició que el sistema tuviera una gran aceptación por parte de los nuevos usuarios. El número de instalaciones en *Bell Laboratories* creció hasta quince, aproximadamente, y su uso también se difundió gradualmente a unas cuantas universidades con propósitos educacionales.



Figura 1.7: pdp 11 de DEC

La primera versión de Unix disponible fuera de *Bell Laboratories* fue la versión 6, en el año 1976. En 1978 se distribuyó la versión 7, que fue adaptada a otros *PDP-11* y a una nueva línea de ordenadores de *DEC* denominada *VAX* (ver figura 1.8). La versión para *VAX* se conocía como *32V*.



Figura 1.8: VAX de DEC

Tras la distribución de la versión 7, Unix se convirtió en un producto y no sólo en una herramienta de investigación o educacional, debido a que el *Unix Support Group (USG)* asumió la responsabilidad y el control administrativo del *Research Group* en la distribución de Unix dentro de *AT&T*.

En el periodo comprendido entre 1977 y 1982, *Bell Laboratories* combinó varios sistemas Unix, de la versión 7 y de la 32v, dando lugar a un único sistema cuyo nombre comercial fue Unix System III. Ésta fue la primera distribución externa desde *USG*.

La modularidad, la sencillez de diseño y el pequeño tamaño de Unix, hicieron que muchas entidades, tales como *Rand*, varias universidades e incluso *DEC*, se pusieran a trabajar sobre él, La Universidad de Berkeley en California desarrolló una variante del sistema Unix para máquinas *VAX*. Esta variante incorporaba varias características interesantes, tales como memoria virtual, paginación por demanda y sustitución de páginas, con lo cual se permitía la ejecución de programas mayores que la memoria física. A esta variante, desarrollada por Bill Joy y Ozlap Babaoglu, se le conoció como *3BSD (Berkeley Software Distributions)*. Todo el trabajo desarrollado por la Universidad de Berkeley para crear *BSD* impulsó a la *Defense Advanced Research Projects Agency (DARPA)* a financiar a Berkeley en el desarrollo de un sistema Unix estándar de uso oficial (*4BSD*). Los trabajos en *4BSD* para *DARPA* fueron dirigidos por expertos en redes y Unix, *DARPA Internet (TCP/IP)*. Este soporte se facilitó de un modo general. En *4.2BSD* es posible la comunicación uniforme entre los distintos dispositivos de la red, incluyendo redes locales (*LAN*), como Ethernet y Token Ring, y extensas redes de ordenadores (*WAN*), como la Arpanet de *DARPA*.

Los sistemas Unix actuales no se reducen a la versión 8, System V o *BSD*, sino que la mayoría de los fabricantes de micro y miniordenadores ofrecen su Unix particular. Así, *Sun Microsystems* los ofrece para sus ordenadores y lo denomina *Solaris*, *Hewlett Packard* lo comercializa con el nombre de *HP-UX*, *IBM* lo implantó en sus equipos *RISC 6000* y lo denomina *AIX*, etc. Con el gran incremento en las prestaciones de los ordenadores personales, también han aparecido versiones para ellos. Dentro de estas nuevas versiones cabe destacar aquellas de distribución libre, como pueden ser *FreeBSD*, *OpenBSD* o el propio Linux, obtienen un alto rendimiento de los procesadores de la familia *80x86* de *Intel* (del *80386* en adelante).

2.2. Aparición de Linux

Linux es un sistema operativo de distribución libre desarrollado inicialmente por Linus Torvalds (ver figura 1.9) en la Universidad de Helsinki (Finlandia). Una comunidad de programadores expertos en Unix, han ayudado en el desarrollo, distribución y depuración de este sistema operativo. El núcleo de Linux no contiene código desarrollado por *AT&T* ni por ninguna otra fuente propietaria. La mayoría del software disponible en Linux ha sido desarrollado por el proyecto *GNU* de la *Free Software Foundation* de Cambridge (Massachusetts). Sin embargo, es toda la comunidad de programadores la que ha contribuido al desarrollo de aplicaciones para este sistema operativo.



Figura 1.9: Linus Torvalds

Con la aparición de ordenadores personales potentes aparece Linux. Inicialmente se trató sólo de un desarrollo llevado a cabo por Linus Torvalds por pura diversión. Linux se basó en *Minix*, un pequeño sistema Unix desarrollado por Andrew S. Tanenbaum.

Los primeros desarrollos de Linux tenían que ver con la conmutación de tareas en el microprocesador 80386 ejecutando en modo protegido, todo ello escrito en lenguaje ensamblador.

No se llevó a cabo ningún anuncio de la versión 0.01 de Linux. Por sí misma, esta versión sólo podía compilarse y ejecutarse en una máquina que tuviese cargado Minix.

El cinco de octubre de 1991 Linus dio a conocer la primera versión “oficial” de Linux, ésta fue la versión 0.02. En este punto Linux podía ejecutar el intérprete de órdenes *bash* (*Bourne Again shell de GNU*) y *gcc* (*el compilador C de GNU*) pero no mucho más. Seguía siendo una versión utilizable solamente por hackers y por personal “cualificado”.

Después de la versión 0.03, Linus pasó a lanzar la versión 0.10, en este punto fue cuando aumentó considerablemente el número de personas que se apuntó al desarrollo del sistema. Después de varias versiones intermedias, Linus incrementó el número y pasó directamente a la versión 0.95 para reflejar sus deseos de que pronto pasaría a ser una versión “oficial” (generalmente al software sólo se le asigna como número de versión la 1.0 cuando se supone que está en su mayoría libre de errores). Esto ocurrió en marzo de 1992. Un año y medio después, a finales de Diciembre de 1993, el núcleo (*kernel*) de Linux estaba en la versión 0.99.pl14, aproximándose asintóticamente a 1.0.

Actualmente Linux es un Unix en toda regla, compatible *POSIX*, capaz de ejecutar *X Window*, *TCP/IP*, *Emacs*, *UUCP* (*Unix to Unix CoPy*), correo electrónico, servicios de noticias, etc.

2.3. Distribuciones de Linux

Una distribución de Linux es una variante de Linux que incorpora determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones hogareñas, empresariales y para servidores. Pueden ser exclusivamente de software libre (por ejemplo: *gobuntu*), o también incorporar aplicaciones o controladores propietarios.

La base de cada distribución incluye el núcleo (*kernel*) Linux, con las bibliotecas y herramientas del proyecto *GNU* y de muchos otros proyectos/grupos de software, como *BSD*, *Xorg*, *Apache*, *MySQL*, *PostgreSQL*, *Perl*, *Python*, *PHP*, *Gnome* y *KDE*.

Antes de que surgieran las primeras distribuciones Linux, un usuario de Linux debía tener algo de experiencia en Unix; no sólo debía conocer qué bibliotecas y ejecutables necesitaba para iniciar el sistema, sino también los detalles importantes que se requerían en la instalación y configuración de los archivos en el sistema.

Las distribuciones de Linux comenzaron a surgir poco después de que el núcleo Linux fuera utilizado por otros programadores además de los creadores originales. Existía mayor interés en desarrollar un sistema operativo que en desarrollar aplicaciones, interfaces para los usuarios o un paquete de software conveniente.

Las distribuciones eran originalmente una cuestión de comodidad para el usuario medio, evitándole la instalación (y en muchos casos la compilación) por separado de paquetes de uso común, pero hoy se han popularizado incluso entre los expertos en éste tipo de sistemas operativos (Unix/Linux).

2.3.1. Principales distribuciones

Dentro de las distribuciones de Linux existe toda una amplia gama de ellas, en este apartado vamos a describir las distribuciones que actualmente son consideradas las más importantes y por tanto las más utilizadas.



Red Hat es la compañía responsable de la creación y mantenimiento de una distribución del sistema operativo GNU/Linux que lleva el mismo nombre: Red Hat Enterprise Linux, y de otra más que lleva el nombre de Fedora.

Red Hat es famoso en todo el mundo por los diferentes esfuerzos orientados a apoyar el movimiento del software libre. No sólo trabajan en el desarrollo de una de las distribuciones más populares de Linux, sino también en la comercialización de diferentes productos y servicios basados en software de código abierto. Asimismo, poseen una amplia infraestructura en la que se cuentan más de quinientos empleados en quince lugares del mundo.

Programadores empleados de Red Hat han desarrollado múltiples paquetes de software libre, los cuales han beneficiado a toda la comunidad. Algunas de las contribuciones más notables ha sido la creación de un sistema de empaquetación de software llamado RPM o Red Hat Package Manager, que es un sistema desarrollado por esta empresa para facilitar la instalación de componentes de Linux, y varias utilidades para la administración y configuración de equipos, como `sndconfig` o `mouseconfig`.

Algunas de las distribuciones basadas en Red Hat Linux más importantes son: Mandriva Linux y Yellow Dog Linux (esta última sólo para PowerPC).



Fedora es una distribución de Linux basada en RPM para propósitos generales, que es soportada por una comunidad internacional de ingenieros, diseñadores gráficos y usuarios que reportan fallos y prueban nuevas tecnologías. Esta distribución Linux cuenta con el respaldo y la promoción de Red Hat.

El proyecto no busca solo incluir software libre y de código abierto, sino ser el líder en ese ámbito tecnológico. Algo que hay que destacar es que los desarrolladores de Fedora prefieren hacer cambios en las fuentes originales en lugar de aplicar los parches específicos en su distribución, de esta forma se asegura que las actualizaciones estén disponibles para todas las variantes de Linux. Max Spevack en una entrevista afirmó que: "Hablar de Fedora es hablar del rápido progreso del software libre y de código abierto".

Durante sus primeras seis versiones se llamó Fedora Core, debido a que solo incluía los paquetes más importantes del sistema operativo. La última versión es Fedora 8, la cual fue liberada el 8 de noviembre de 2007.



SuSE Linux es una de las más conocidas distribuciones Linux existentes a nivel mundial, se basó en sus orígenes en Slackware. Entre las principales virtudes de esta distribución se encuentra el que sea una de las más sencillas de instalar y administrar, ya que cuenta con varios asistentes gráficos para completar diversas tareas en especial por su gran herramienta de instalación y configuración *YasT*.

Su nombre "SuSE" es el acrónimo, en alemán "*Software und Systementwicklung*", el cual formaba parte del nombre original de la compañía y que se podría traducir como "desarrollo de software y sistemas". El nombre actual de la compañía es SuSE Linux, habiendo perdido el primer término su significado (al menos oficialmente).

El cuatro de noviembre de 2003, la compañía multinacional estadounidense Novell anunció que iba a comprar SuSE Linux. La adquisición se llevó a cabo en enero de 2004. En el año 2005, en la Linux World, Novell, siguiendo los pasos de Red Hat Inc., anunció la liberación de la distribución SuSE Linux para que la comunidad fuera la encargada del desarrollo de esta distribución, que ahora se denomina openSuSE.

El cuatro de agosto de 2005, el portavoz de Novell y director de relaciones públicas Bruce Lowry anunció que el desarrollo de la serie SuSE Professional se convertiría en más abierto y entraría en el intento del proyecto de la comunidad openSuSE de alcanzar a una audiencia mayor de usuarios y desarrolladores. El software, por la definición de código abierto, tenía ya su código fuente "abierto", pero ahora el proceso de desarrollo sería más "abierto" que antes, permitiendo que los desarrolladores y usuarios probaran el producto y ayudaran a desarrollarlo.

Anteriormente, todo el trabajo de desarrollo era realizado por SuSE, y la versión 10.0 fue la primera versión con una beta pública. Como parte del cambio, el acceso en línea al servidor YaST de actualización sería complementario para los usuarios de SuSE Linux, y siguiendo la línea de la mayoría de distribuciones de código abierto, existiría tanto la descarga gratuita disponible mediante web como la venta del sistema operativo en caja. Este cambio en la filosofía condujo al lanzamiento de SuSE Linux 10.0 el seis de octubre de 2005 en "*OSS, Open Source Software*" (código completamente abierto).



Mandriva Linux (antes Mandrakelinux y Mandrake Linux) es una distribución Linux aparecida en julio de 1998 propiedad de Mandriva, enfocada hacia usuarios sin experiencia en este mundo que buscan sencillez y un uso sin problemas.

Se distribuye mediante la licencia: Licencia pública general (*GPL*) de *GNU*, y es posible descargar su distribución en formato ISO, sus asistentes o sus repositorios.

La primera edición se fundamentó en Red Hat Linux (versión 5.1) y escogió el entorno gráfico de *KDE (K Desktop Environment)*. Desde entonces ha seguido su propio camino, separado de Red Hat y ha incluido numerosas herramientas propias o modificadas, fundamentalmente dirigidas a facilitar la configuración del sistema. Mandrake (su anterior nombre) también es conocida por compilar sus paquetes con optimizaciones para procesadores Pentium y superiores, incompatibles con versiones más antiguas tales como 386 y 486.



Debian o Proyecto Debian es una comunidad conformada por desarrolladores y usuarios, que pretende crear y mantener un sistema operativo *GNU* basado en software libre precompilado y empaquetado, en un formato sencillo en múltiples arquitecturas de computador y en varios núcleos.

Debian nace como una apuesta por separar en sus versiones el software libre del software no libre. El modelo de desarrollo del proyecto es ajeno a motivos empresariales o comerciales, siendo llevado adelante por los propios usuarios, aunque cuenta con el apoyo de varias empresas en forma de infraestructuras. Debian no vende directamente su software, lo pone a disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuir comercialmente este software mientras se respete su licencia.

La comunidad de desarrolladores de Debian cuenta con la representación de software *in the Public Interest*, una organización sin ánimo de lucro que da cobertura legal a varios proyectos de software libre.

La primera adaptación del sistema Debian, sino también la más desarrollada, es Debian GNU/Linux, basada en el núcleo Linux, y como siempre utilizando herramientas de *GNU*. Existen también otras adaptaciones con diversos núcleos: *Hurd* (Debian GNU/Hurd); *NetBSD* (Debian GNU/NetBSD) y *FreeBSD* (Debian GNU/kFreeBSD).



GnuLinEx es una distribución de software libre que incluye el núcleo de Linux y está basada en Debian GNU/Linux y *GNOME* (*GNU Network Object Model Environment*), contando con OpenOffice.org como Suite Ofimática, entre otras aplicaciones.

Está impulsado por la Consejería de Infraestructuras y Desarrollo Tecnológico de la Comunidad Autónoma de Extremadura (España), siendo pionero y secundado por otros organismos públicos y privados del resto de España. Durante un periodo considerable de tiempo, la comunidad extremeña ofreció también apoyo a la de Andalucía (la cual se inspiró en GnuLinEx para desarrollar Guadalinux) en la implantación de soluciones abiertas en colegios, administración, etc.

El diecinueve de junio de 2006 se liberó GnuLinex 2006, con Gnome 2.14.1, Xorg 6.9, y núcleo de Linux 2.6.16.



Ubuntu es una distribución Linux que ofrece un sistema operativo predominantemente enfocado a computadoras de escritorio aunque también proporciona soporte para servidores.

Basada en Debian GNU/Linux, Ubuntu concentra su objetivo en la facilidad de uso, la libertad de uso, los lanzamientos regulares (cada seis meses) y la facilidad en la instalación. Ubuntu es patrocinado por Canonical Ltd., una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth.

El nombre de la distribución proviene del concepto *zulú* y *xhosa* de Ubuntu, que significa "humanidad hacia otros" o "yo soy porque nosotros somos". Ubuntu es un movimiento sudafricano encabezado por el obispo Desmond Tutu, quien ganó el Premio Nobel de la Paz en 1984 por sus luchas en contra del apartheid en Sudáfrica. El sudafricano Mark Shuttleworth, mecenas del proyecto, se encontraba muy familiarizado con la corriente. Tras ver similitudes entre los ideales de los proyectos *GNU*, Debian y en general con el movimiento del software libre, decidió aprovechar la ocasión para difundir los ideales de Ubuntu. El eslogan de Ubuntu "Linux para seres humanos" (en inglés "Linux for Human Beings") resume una de sus metas principales: Hacer de Linux un sistema operativo más accesible y fácil de usar.

3. Software libre y Linux

En esta sección trataremos de explicar e introducir lo que se persigue con la filosofía del software libre y todos los términos o elementos que dicha filosofía implica. Porque aunque el software libre está presente cada vez más en los medios de comunicación, en las conversaciones de los profesionales de la informática, e incluso empieza a estar en boca de los ciudadanos en general, aún es desconocido por muchos de nosotros.

El software libre tuvo sus inicios en pequeños grupos de entusiastas y activistas que querían cambiar la industria del software. Después de muchos años esta comunidad ha crecido rápidamente e incorporado no solamente a voluntarios en todo el mundo, sino que también ha atraído la atención y la colaboración de centenares de empresas claves.

Inicialmente el software libre fue popular en los servidores y con el paso del tiempo y el trabajo de miles de voluntarios las piezas faltantes se completaron. Linux que antes estaba limitado a ser un sistema que no era visible a los usuarios finales (Por ejemplo, Google y Amazon son sistemas construidos sobre Linux) ahora es un sistema que es usado por miles de usuarios en todo el mundo en sus computadoras personales, teléfonos u organizadores personales.

El software libre es propiedad de todos: Cada persona en el mundo tiene derecho a usar el software, modificarlo y copiarlo de la misma manera que los autores de este mismo. Es un legado de la humanidad que no tiene propietario, de la misma manera que las leyes básicas de la física o las matemáticas. No existe un monopolio y no es necesario pagar peaje por su uso.

3.1. GNU

Con la explosión de la microinformática, el descenso de precio de los sistemas informáticos y su popularización entre las empresas, aparecieron las primeras compañías de software. Muchas de ellas empezaron contratando *hackers* que estaban alrededor de los centros de cálculo de las universidades, de forma que éstas se fueron despoblando de aquellos pioneros. Pero, además, muchas de estas empresas creyeron que si denegaban el acceso a los usuarios y a otros desarrolladores al código fuente de las aplicaciones que mejoraban o desarrollaban, podrían realmente conseguir una ventaja competitiva. Éste fue un punto de inflexión importante, ya que se rompió con la tradición que había imperado hasta entonces de compartir el código.

Poco a poco se fue extendiendo un modelo de código cerrado en el cual el software se vendía sin el código fuente y, cada vez más, las libertades de los usuarios se fueron acortando. Esta fue la época en que aparecieron técnicas como, por ejemplo, las bombas de tiempo (aplicaciones Trial) que limitaban el periodo durante el cual un usuario podía utilizar un producto. Los programas *shareware* popularizarían más tarde estas bombas de tiempo como sistema para obligar a los usuarios a adquirir una licencia.

Una de las personas que había vivido de cerca toda aquella evolución era Richard Stallman (ver figura 1.10), quien fue pionero en defender las libertades que se habían perdido y acuñó el término 'software libre'. El 27 de septiembre de 1983 Richard Stallman muy preocupado por esta pérdida de libertades anunciaba en el foro *Usenet net-unix.wizards*, que empezaba a trabajar sobre una implementación libre de un sistema inspirado en Unix que denominaría *GNU* y que estaría libre de código de *AT&T*, es decir, una implementación desde cero sin posibles problemas legales con *AT&T*. En el mensaje a *Usenet*, Stallman explicaba detalladamente su experiencia como desarrollador de sistemas y pedía la ayuda de todo el mundo que quisiera ofrecer parte de su tiempo, dinero o hardware.



Figura 1.10: Richard Stallman

El 1984, Stallman creó la *Free Software Foundation* con el objetivo de crear el sistema Unix libre *GNU* y la potenciación del software libre. La definición de software libre propuesta por la *Free Software Foundation*, se basa en cuatro libertades básicas que cualquier programa considerado libre debe proporcionar:

- Libertad para utilizar el programa para cualquier propósito.
- Libertad para poder estudiar cómo funciona el programa. Lo cual implica acceso al código fuente del mismo.
- Libertad para redistribuir el programa.
- Libertad para hacer modificaciones y distribuir las mejoras. Lo cual implica también acceso al código fuente del mismo.

El software libre se basa en la cooperación y la transparencia y garantiza una serie de libertades a los usuarios. Estos aspectos, junto al hecho de que su desarrollo ha sido paralelo al de Internet, han causado que sea abanderado para un gran número de usuarios que tienen una concepción libertaria del uso de las nuevas tecnologías. Los programas que no son libres se les llaman propietarios o privativos. Por ejemplo, todas las versiones de Microsoft Windows o Adobe Acrobat son ejemplos de software propietario.

Durante los años 80 Stallman continuó trabajando en el desarrollo de las herramientas necesarias para crear un sistema operativo completamente libre. Publicó una versión del editor *GNU Emacs* y trabajó en herramientas que son fundamentales para el movimiento del software libre, como, por ejemplo, el compilador *GCC (GNU Compiler Collection)* o el depurador *GDB (GNU DeBugger)*.

Ya en sus inicios Stallman identificó la necesidad de crear las protecciones jurídicas necesarias para el software libre. En 1989 publicó la versión 1.0 de la licencia *GPL (General Public License)* un proyecto que elaboraba desde 1985 y que consistía en un contrato entre el autor del software y el usuario que garantizaba la cesión de los derechos que definían al software libre. La licencia *GPL* era una herramienta legal muy importante dado que Stallman había padecido mucho viendo cómo algunos programadores cogían código que era software libre, hacían modificaciones y no aportaban estas modificaciones a la comunidad.

3.2. Libertad y coste

Es habitual que los usuarios confundan el software libre con el software gratuito. Es importante distinguir entre las libertades que nos proporciona un software y el coste del mismo.

Un programa, por el simple hecho de ser gratuito, no es ni mucho menos libre. Por ejemplo, Internet Explorer de Microsoft es un programa gratuito pero no es libre, ya que no da a sus usuarios la posibilidad de estudiarlo (incluyendo el acceso a su código fuente), ni de mejorarlo, ni de hacer públicas estas mejoras con el código fuente correspondiente, de manera que todo el mundo se pueda beneficiar. Internet Explorer es un programa propietario en cuanto a las libertades y gratuito en cuanto al coste.

Existe una distinción fundamental entre los programas que garantizan los derechos de distribución y modificación, el software libre, y los que no los garantizan que consideramos software propietario.

Respecto al coste, cualquier software libre se puede vender, siempre y cuando se respeten las libertades originales que lo definen. Por ejemplo, la empresa francesa Mandrake o la norteamericana Novell venden distribuciones de GNU/Linux, y se trata de software libre porque conserva las libertades que lo definen.

3.3. Open Source

Durante el año 1998, Eric S. Raymond, Bruce Perens y otros *hackers* involucrados en el desarrollo de software libre lanzaron la *Open Software Initiative* y propusieron el uso del término *Open Source* (código abierto) en contraposición al término *free software* (software libre) como término más atractivo al entorno empresarial. El término *free software* en el mundo anglófono (de habla inglesa) creaba una situación incómoda debido a la doble acepción que en inglés tiene el término *free* (que puede significar gratuito o libre). La gran mayoría de empresas en Estados Unidos usan principalmente el término código abierto para evitar dar la percepción que el software libre es un recurso totalmente gratuito y para poner énfasis en el valor diferencial que representa el hecho de que el código fuente está disponible.

Bruce Perens, de la *Open Source Initiative* y antiguo coordinador de la distribución de Linux Debian, creó una lista (ver tabla 1.1) de condiciones que debe cumplir un programa para poder ser considerado *Open Source*. Estas condiciones son muy similares y, de hecho están basadas, en las directrices de software libre de Debian.

Nº	Condición	Descripción
1	Libre distribución	No se puede impedir la venta o distribución del programa o parte de él. Así mismo, tampoco se puede exigir el pago de una tasa a cambio de su distribución por parte de terceros.
2	Código fuente	El programa debe incluir su código fuente y no se puede restringir su redistribución.
3	Trabajos derivados	No debe impedirse realizar modificaciones o trabajos derivados del programa y debe permitirse que éstos sean distribuidos bajo los mismos términos del software original.
4	Integridad del código fuente original	Puede exigirse que una versión modificada del programa tenga un nombre y número de versión diferente que el programa original para poder proteger al autor original de la responsabilidad de estas versiones.
5	No discriminación contra personas o grupos	Las condiciones de uso del programa no pueden discriminar contra una persona o un grupo de personas.

N°	Condición	Descripción
6	No discriminación contra usos	No se puede negar a ninguna persona hacer uso del programa para ningún fin como, por ejemplo, comercial o militar.
7	Distribución de la licencia	Los derechos del programa deben aplicarse a todos quienes se redistribuyen el programa sin ninguna condición adicional.
8	La licencia no debe ser específica de un producto	Los derechos garantizados al usuario del programa no deben depender de que el programa forme parte de una distribución o paquete particular de software.
9	La licencia no debe restringir otro software	La licencia no debe poner restricciones en otros programas que se distribuyen junto con el software licenciado.
10	La licencia no debe ser tecnológicamente neutra	No puede existir ninguna disposición de la licencia que obligue al uso de una tecnología concreta.

Tabla 1.1: Lista de condiciones que debe cumplir un programa para ser considerado *Open Source*

Estas condiciones también son aplicables a cualquier programa que sea software libre y pueden ayudarnos a matizar sus implicaciones.

3.4. Licencias en el software libre

Lo que diferencia al software libre del resto del software es un aspecto legal: La *licencia*, esta se trata de un *contrato* entre el autor o propietario de los derechos y los usuarios; que estipula lo que éstos pueden hacer con su obra: Uso, redistribución, modificación, etc., y en qué condiciones, Es decir, la licencia contiene las normas de uso a las que han de atenerse los usuarios, los distribuidores, los integradores y otras partes implicadas en el mundo de la informática.

Las condiciones y/o restricciones que imponen las licencias sólo pueden ser precisadas por los propios autores, que según la normativa de propiedad intelectual son los propietarios de la obra. La propiedad de la obra será de los autores, ya que la licencia no supone transferencia de propiedad, sino solamente derecho de uso y, en algunos casos, de distribución. Es necesario saber que cada nueva versión de un programa es considerada como una nueva obra. El autor tiene, otra vez, plena potestad para hacer con su obra lo que le apetezca, incluso distribuirla con términos y condiciones totalmente diferentes, es decir, una licencia diferente a la anterior.

Partiendo de todo lo dicho, vamos a centrarnos en el análisis de diversas licencias.

3.4.1. Licencias tipo *BSD*

La licencia *BSD* (*Berkeley Software Distribution*) tiene su origen en la publicación de versiones de Unix realizadas por la universidad californiana de Berkeley, en EE.UU. La única obligación que exige es la de dar crédito a los autores, mientras que permite tanto la redistribución binaria, como la de los códigos fuentes, aunque no obliga a ninguna de las dos en ningún caso. Asimismo, da permiso para realizar modificaciones y ser integrada con otros programas casi sin restricciones.

La licencia *BSD* es ciertamente muy popular. Estas licencias reciben el nombre de minimalistas, ya que las condiciones que imponen son pocas, básicamente asignar la autoría a los autores originales. Su concepción se debe al hecho de que el software publicado bajo esta

licencia era software generado en universidades con proyectos de investigación financiados por el gobierno de los Estados Unidos.

Entre las licencias de tipo *BSD* podemos encontrar: La de *X Window*, *Tcl/Tk* y *Apache*. La mayoría de ellas son una copia calcada de la original de Berkeley, modificando todo lo referente a la autoría. Otras, como la *Apache*, incluyen alguna cláusula adicional, como la imposibilidad de llamar las versiones redistribuidas de igual manera. Todas suelen incluir, como ella, la prohibición de usar el nombre del propietario de los derechos para promocionar productos derivados.

Asimismo, todas las licencias, sean de tipo *BSD* o no, incluyen una *limitación de garantía* que es en realidad una *negación de garantía*, necesaria para evitar demandas legales por garantías implícitas. Aunque se ha criticado mucho esta negación de garantía en el software libre, es práctica habitual en el software propietario, que generalmente sólo se garantiza que el soporte es correcto y el programa en cuestión se ejecuta.

3.4.2. La licencia Pública General de GNU (*GNU GPL*)

La *Licencia Pública General del proyecto GNU*, más conocida por su acrónimo en inglés *GPL*, es la licencia más popular y conocida de todas las licencias del mundo del software libre. Su autoría corresponde a la *Free Software Foundation* (promotora del proyecto *GNU*) y en un principio fue creada para ser la licencia de todo el software generado por la *Free Software Foundation*. Sin embargo, su utilización ha ido más allá hasta convertirse en la licencia más utilizada (más del 70% de los proyectos anunciados en *FreshMeat* están licenciados bajo la *GPL*), incluso por proyectos bandera del mundo del software libre, como es el caso del núcleo Linux.

La licencia *GPL* permite la redistribución binaria y la de las fuentes, aunque, en el caso de que redistribuya de manera binaria, obliga a que también se pueda acceder a las fuentes. Asimismo, está permitido realizar modificaciones sin restricciones, aunque sólo se pueda integrar código licenciado bajo *GPL* con otro código que se encuentre bajo una licencia idéntica o compatible, lo que ha venido a llamarse el efecto *viral* de la *GPL*, ya que el código publicado una vez con esas condiciones nunca puede cambiar de condiciones.

La licencia *GPL* está pensada para asegurar la libertad del código en todo momento, ya que un programa publicado y licenciado bajo sus condiciones nunca podrá ser hecho propietario. Es más, ni ese programa ni modificaciones al mismo pueden ser publicados con una licencia diferente a la propia *GPL*.

También incluye *negaciones de garantía* para proteger a los autores. Asimismo, para proteger la buena fama de los autores originales, toda modificación de un fichero fuente debe incluir una nota con la fecha y autor de cada modificación.

3.5. Licencias de otros recursos libres

Las licencias de software libre han sido fuente de inspiración para otros recursos intelectuales, de tal modo que muchos de ellos las han adoptado de manera directa, especialmente en el caso de la documentación o la fotografía, en otros casos han sido adaptadas ligeramente, como es el caso de la pionera *Open Audio License*.

3.5.1. Licencia de documentación libre de GNU

Después de darse cuenta que un documento no es lo mismo que un programa, Richard Stallman promovió una licencia para los documentos que acompañan a los programas y para otros documentos de carácter técnico o didáctico.

Una de las preocupaciones de la licencia es reconocer la autoría e impedir que se tergiversen ideas u opiniones expresadas por el autor. Para ello, se exige que las obras derivadas exhiban en la portada un título distinto a los de las versiones anteriores (salvo permiso expreso) y se nombre expresamente en dónde se puede conseguir el original. También deben listarse como autores los más importantes de los originales además de los autores de las modificaciones y deben conservarse todas las notas sobre derechos de autor. Asimismo, deben conservarse agradecimientos, dedicatorias, así como respetar el apartado de historia, si lo tiene, añadiendo las modificaciones nuevas. Incluso pueden nombrarse secciones invariantes y textos de cubiertas, que nadie puede modificar ni eliminar.

3.5.2. Licencias de *Creative Commons*



En el 2001 se fundó *Creative Commons*, dirigido por expertos en propiedad intelectual, derecho en la sociedad de la información, e informática, con el propósito de fomentar la existencia, conservación y accesibilidad de recursos intelectuales cedidos a la comunidad de diversas maneras. Uno de sus proyectos más conocidos fue el desarrollo, a finales del 2002 de una serie de licencias concebidas, no para software, sino para trabajos literarios, artísticos, didácticos, etc. Su característica más sobresaliente, además de estar avaladas por profesionales del derecho, es que permiten al autor seleccionar qué tipo de libertades cede, además de la de copia, según cuatro dimensiones: Dar crédito al autor original, permitir trabajos derivados, permitir redistribución comercial y permitir cambiar la licencia. Así, por ejemplo, la licencia de los cursos del MIT (MIT Open Courseware License Version 1.0) está basada en la de *Creative Commons* que obliga a dar crédito, impide el uso comercial y obliga a conservar la licencia en trabajos derivados.

Poner obras bajo una licencia *Creative Commons* no significa que no tengan copyright. Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones. ¿Qué condiciones? bajo las condiciones mostradas en la siguiente lista las cuales podemos escoger o unir según nuestra conveniencia.



Reconocimiento (Attribution): El material creado por un artista puede ser distribuido, copiado y exhibido por terceras personas si se muestra en los créditos.



No comercial (Non Commercial): El material original y los trabajos derivados pueden ser distribuidos, copiados y exhibidos mientras su uso no sea comercial.



Sin obra derivada (No Derivate Works): El material creado por un artista puede ser distribuido, copiado y exhibido pero no se puede utilizar para crear un trabajo derivado del original.



Compartir igual (Share Alike): El material creado por un artista puede ser modificado y distribuido pero bajo la misma licencia que el material original.

Hay un total de seis licencias *Creative Commons* (ver tabla 1.2) para escoger. Las cuales están formadas por distintas combinaciones de las cuatro condiciones mostradas en el apartado anterior. Estas seis licencias son:







Simbología	Descripción
	Reconocimiento: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos.
	Reconocimiento – Sin obra derivada: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se pueden realizar obras derivadas.
	Reconocimiento – Sin obra derivada – No comercial: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial. No se pueden realizar obras derivadas.
	Reconocimiento – No comercial: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial.
	Reconocimiento – No comercial – Compartir igual: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.
	Reconocimiento – Compartir igual: El material creado por un artista puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. Las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.

Tabla 1.2: Los seis tipos de licencias generadas con combinaciones de las cuatro condiciones

Hemos podido observar la importancia que tienen las licencias dentro del mundo del software libre y de los demás recursos libres. Así como también presentamos algunas de las licencias más importantes que existen dentro de la gran variedad de licencias del software libre, su motivación, sus repercusiones y sus ventajas e inconvenientes.

En definitiva, podemos decir que la *GPL* trata de maximizar las libertades que tiene el usuario del software (lo reciba directamente de su autor o no), mientras que las licencias tipo *BSD* lo que hacen es maximizar las libertades del modificador o redistribuidor.

4. Introducción a la administración

Cada sistema debe tener su propio administrador o persona encargada de que todo esté a punto en cada momento. Esta labor requiere una serie de conocimientos que los usuarios finales no necesitan dominar. Además, es necesario invertir un tiempo considerable para estos menesteres.

La administración del sistema es uno de los aspectos menos estándar de un sistema tipo Unix. Tanto las órdenes empleadas como los archivos de configuración pueden variar de unos sistemas a otros. Hay que señalar que el mejor aliado de cualquier administrador que se precie de serlo es el manual (*man*) del sistema, donde podemos encontrar todas las peculiaridades de nuestro sistema concreto que nos ayudarán a resolver cualquier tipo de problema.

4.1. Ciclo de vida del sistema

Un sistema informático pasa por varias etapas a lo largo de su vida (ver figura 1.11). Desde el punto de vista del administrador del sistema, cada etapa queda caracterizada por un conjunto distinto de actividades que es necesario llevar a cabo.

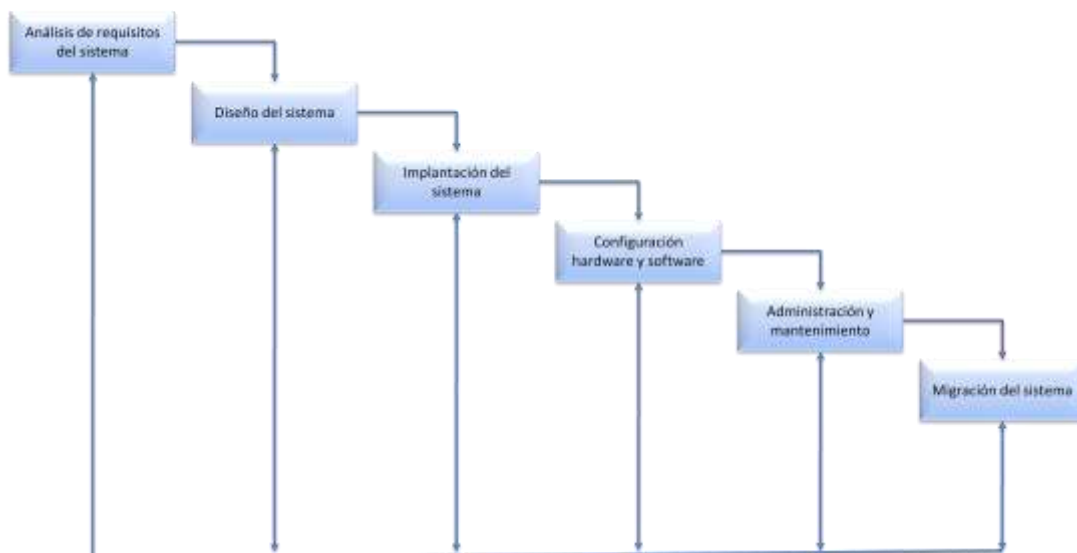


Figura 1.11: Ciclo de vida del sistema

Análisis de requisitos del sistema

Se establecen qué problemas tiene que solucionar el sistema informático, a qué actividades de la organización debe dar soporte y qué tipo de servicios debe prestar. El resultado es un documento de requisitos que recoge todos los aspectos mencionados anteriormente.

Diseño del sistema

Una vez conocidos los requisitos, se analiza qué componentes hay que utilizar para satisfacer dichos requisitos. Los componentes generalmente son de tipo: Hardware y software.

Implantación del sistema

Consiste en montar, instalar y adaptar los componentes hardware y software, según el documento de diseño, para que el sistema informático satisfaga una serie de requisitos. Cada componente se instala según las instrucciones dadas por el proveedor del componente.

Configuración hardware y software de forma que el sistema cumpla los requisitos exigidos

Una vez instalados los componentes es necesario adaptarlos a las necesidades específicas del sistema. Una vez configurados todos los componentes, éstos proporcionarán los servicios tal y como se especificó en el documento de requisitos.

Administración y mantenimiento (explotación)

El sistema se encuentra ya en funcionamiento y prestando los servicios para los que fue creado. Durante todo el tiempo de servicio será necesario mantener actualizado el software para evitar errores y problemas de seguridad, funcionalidades, ajustar parámetros de rendimiento, etc.

Migración, desmantelamiento del sistema

Si el sistema queda obsoleto, será necesaria la implantación de uno nuevo. Esta etapa asegura que se podrá reutilizar, a ser posible, la totalidad de los datos y hacer que la migración hacia el nuevo sistema se haga de forma progresiva, reduciendo al mínimo el tiempo en el que el sistema se encuentra inoperativo.

La administración de sistemas es una actividad muy amplia que se centra fundamentalmente en los puntos cuatro y cinco del ciclo de vida de un sistema informático, aunque en la realidad abarca más puntos.

4.2. El administrador del sistema

Los sistemas tipo Unix diferencian entre los distintos usuarios, de manera que se regula qué es lo que podemos hacerle a otros usuarios o al propio sistema. Cada uno de ellos tiene su propia cuenta, la cual incluye nombre de conexión, grupo al que pertenece, directorio de arranque, etc. De todas las cuentas del sistema, sin duda alguna la más importante es la denominada cuenta de administrador o superusuario, cuyo nombre de conexión es *root*. Esta cuenta es siempre creada automáticamente en la instalación de cualquier sistema tipo Unix, momento en que se establece una palabra clave inicial. Es un aspecto clave en el mantenimiento de la seguridad informática asegurar la confidencialidad de la clave del administrador.

Normalmente las cuentas de usuarios tienen asociadas una serie de restricciones, de forma que nadie pueda molestar al resto, a lo sumo a ellos mismos. Nadie va a poder borrar directorios como */etc* o */bin*, ni nadie va a poder desactivar una impresora. Todo este tipo de restricciones no son aplicables al administrador (*root*). El administrador tiene plenos poderes para borrar, crear o modificar cualquier archivo o directorio del sistema, para ejecutar programas especiales o para dar formato al disco. Como *root* puede hacer todo lo que desee, es necesario que extreme sus precauciones, ya que si no es así, las consecuencias pueden ser catastróficas.

Normas para prevenir los accidentes cuando estamos conectados como administradores del sistema:

- Después de teclear una orden y antes de pulsar la tecla *Enter*, verificar las consecuencias que pueden producirse. Por ejemplo, antes de borrar un directorio, releer la orden con objeto de comprobar que todo es correcto.
- Evitar conectarse como *root* a no ser que sea estrictamente necesario.
- Utilizar un *prompt* diferente para la cuenta de *root*. Lo más habitual es emplear como *prompt* el carácter #.

4.3. La primera regla del administrador

La “primera regla” del administrador de sistemas consiste en proporcionar y mantener acceso a los recursos del sistema. Independientemente de la plataforma informática de que se trate, todos los sistemas operativos proporcionan mecanismos para manipular recursos. Entre estos recursos se encuentran los archivos, las aplicaciones, los periféricos, el ancho de banda, los ciclos de la CPU, la memoria y el espacio de almacenamiento.

La identificación de los recursos, de sus propietarios y sus usuarios, a la vez que la definición de las formas de acceso y las autoridades relacionadas, constituye un proceso de análisis de los requisitos. Los administradores de sistemas transforman esta generalización en especificaciones de cada sistema, de sus usuarios y de su gestión.

El mantenimiento del acceso a los recursos de un sistema trasciende el mantenimiento del propio sistema. Si éste falla o se vuelve inaccesible por cualquier causa, el administrador de sistemas deberá restaurar prontamente el funcionamiento normal del mismo. La mayoría piensa que haciendo copias de seguridad se soluciona este problema. Sin embargo, el administrador de sistemas deberá girar más bien en torno a la planificación de la recuperación, lo cual suele implicar tanto la realización de copias de seguridad, como la valoración de riesgos y pruebas.

A lo largo del funcionamiento normal y, especialmente, en la medida en que las organizaciones crecen y se van adaptando al medio, el uso de los recursos también va cambiando. Cuando el uso de cualquier recurso excede a su capacidad, probablemente esto se verá como un desastre. El administrador de sistemas habrá fallado en su primera regla de proporcionar el acceso necesario. Un fallo también puede causar dificultades de otros servicios. Por ejemplo, si los registros del sistema ocupan toda la partición de un disco, esto puede evitar el envío de correo electrónico a través de ese sistema.

La seguridad es el corolario de la primera regla, que es la de denegar acceso a los recursos del sistema, así como la de garantizar que los servicios se prestan correctamente, incluso durante y después de los ataques de los intrusos. La seguridad implica tanto la realización de las normas como su cumplimiento.

Se deben habilitar las normas apropiadas de acceso, en la medida en que los propietarios de cada recurso lo autorizan. Los procesos de definición y exigencia de estas normas constituyen los dos elementos esenciales de la seguridad de los sistemas.

El análisis de los requisitos, la planificación de la recuperación y la seguridad constituyen el eje principal del trabajo del administrador de sistemas.

4.4. Responsabilidades del administrador

El administrador del sistema o superusuario tiene una serie de responsabilidades que pueden ser divididas en tres grupos: Responsabilidades hardware, software y responsabilidades con los usuarios.

Responsabilidades hardware

- Verificar la correcta instalación del hardware.
- Comprobar el estado de los periféricos y ser capaz de buscar el fallo en caso de errores de la instalación.
- Instalar nuevos dispositivos hardware (memoria, discos, terminales, etc.).

- Determinar limitaciones en los dispositivos que puedan comprometer la prestación de servicios con la calidad necesaria.

Responsabilidad software

La responsabilidad sobre el *mantenimiento del software* es cada vez más importante puesto que a medida que se emplean sistemas para proporcionar servicios complejos, el software se hace cada vez más difícil de mantener.

Dentro de las responsabilidades del mantenimiento software podemos hacer una clasificación adicional entre software del sistema y software específico. El software del sistema es aquel que proporciona los servicios básicos de funcionamiento de un sistema tipo Unix. Por ejemplo, el software que permite a los usuarios conectarse al sistema o el propio sistema operativo. El software específico se refiere a aquel que proporciona un servicio determinado utilizando como plataforma nuestro sistema operativo tipo Unix, como por ejemplo servidores de bases de datos o servidores web.

Responsabilidades derivadas del software del sistema

- Instalar el sistema operativo, configurarlo y mantenerlo al día con las actualizaciones oportunas.
- Crear y mantener los sistemas de archivos, detectando y corrigiendo los posibles errores que puedan producirse.
- Controlar la utilización de este sistema de archivos y su crecimiento.
- Diseñar e implementar las rutinas para realizar copias de seguridad, así como para su posterior recuperación.
- Configurar y mantener el software de cualquier dispositivo: Impresoras, módem, tarjetas de red, etc.
- Actualizar el sistema operativo en caso de poseer una versión más moderna.
- Instalar el software de cualquier aplicación (*X Window*, bases de datos, procesadores de texto, etc.).

Responsabilidades derivadas del software específico

- Instalación y configuración inicial del software.
- Evaluación en las repercusiones en la seguridad global del sistema.
- Labores de administración específicas del servicio prestado.

Responsabilidades sobre los usuarios

- Añadir nuevos usuarios y dar de baja a los que ya no se conectan al sistema. Esto cobra especial relevancia cuando existen políticas de acceso con fines económicos.
- Permitir el acceso a los usuarios de forma controlada.

- Evaluar las necesidades en cuanto a equipos se refiere. Determinar si es necesario añadir nuevos discos, impresoras, memorias, etc. con objeto de que los usuarios encuentren un entorno agradable de trabajo.
- Proporcionar asistencia a cada una de las personas.
- Tener a los usuarios informados en todo momento de los posibles nuevos servicios y sus características. También es necesario que los usuarios conozcan las políticas de seguridad y de prestación de servicios, de forma que el uso de los sistemas se haga siempre dentro del marco legal de cada país.

Aspectos éticos de la administración de sistemas

- Respeto a la privacidad sobre todas las cosas. Como administrador de sistemas se dispone de la capacidad para ver y hacer cualquier cosa sobre los datos y programas de los usuarios. Este hecho no debe implicar una posición de poder, sino de responsabilidad.
- Pueden existir sistemas con políticas que permitan conocer en todo momento qué está haciendo un usuario y de qué forma está haciendo uso del servicio prestado por el sistema informático. En este caso el usuario debe ser informado de las medidas de inspección que se pueden llevar a cabo sobre sus datos y sus actividades.
- Las actividades de administración de un sistema informático deben llevarse a cabo con la máxima profesionalidad y seriedad.

4.5. Seguridad en la administración

El administrador es el responsable de mantener una política de seguridad en el sistema. Esta política de seguridad puede implicar diversas acciones, las cuales incluyen desde comprobar que no existen agujeros en la seguridad hasta detectar que nadie pierde el tiempo.

Todo administrador debe tener siempre presente los siguientes aspectos relacionados con la seguridad:

- El administrador del sistema tiene acceso sin restricciones a todos los recursos. Si un administrador no es consciente de lo anterior, posiblemente sea él mismo el que tire el sistema abajo sin necesidad de ningún tipo de ayuda externa, es decir, que no serán necesarios agentes externos que causen el caos dentro del sistema ya que será él mismo el que los cause.
- Es muy peligroso emplear privilegios de administrador por periodos prolongados de tiempo. Los errores causados por el incumplimiento de esta norma de seguridad pueden tener consecuencias fatídicas sobre el sistema.
- Los usuarios deben emplear contraseñas adecuadas. Es aconsejable por parte del administrador buscar posibles cuentas de usuarios sin contraseña. La idea es que en ocasiones resulta útil ponerse en el papel de quienes puedan atentar contra la seguridad del sistema con objeto de conocer los puntos débiles de nuestro sistema.
- La palabra clave del administrador debe mantenerse estrictamente en secreto y ser conocida como máximo por dos o tres usuarios. Esta palabra clave debe ser modificada periódicamente.

- Vigilar la cantidad de accesos erróneos producidos en el sistema, los cuales quedan normalmente apuntados en un archivo de registro.
- Los directorios del sistema, tales como */etc*, */bin*, */dev*, etc., no deben tener permiso de escritura para los usuarios ordinarios.
- El acceso al terminal que actúa como consola, así como a los terminales donde se puede acceder como *root*, deben estar restringidos. Dicho de otro modo, sólo debe ser posible conectarse como administrador del sistema desde aquellos terminales que se consideren seguros.
- La política de seguridad debe estar perfectamente definida siempre que los mecanismos de seguridad del sistema tipo Unix lo permita.
- Vigilar estrechamente a los usuarios potencialmente peligrosos. Ciertos usuarios pueden dedicar cantidades ingentes de tiempo con el propósito de romper la seguridad del sistema.
- Eliminar de la variable *PATH* del administrador el directorio actual. Un buen *PATH* podría ser el siguiente: *PATH=/etc:/bin:/usr/bin*.
- No relajar las políticas de seguridad porque estas constituyan un problema de administración. En ocasiones los administradores de sistemas se pueden ver tentados a autorizar ciertas operaciones potencialmente peligrosas, porque autorizarlas es más fácil o rápido que buscar una solución segura.
- Consultar periódicamente la información sobre fallos de seguridad informática que se publican en Internet.
- Aplicar cuanto antes las correcciones de seguridad que vayan publicando los proveedores del software de nuestro sistema.

Siguiendo todas las normas citadas anteriormente no conseguiremos que nuestro sistema sea inexpugnable, pero la falta de cumplimiento de dichas normas asegura que nuestro sistema tiene agujeros. La seguridad es un aspecto fundamental que debe tener en cuenta todo administrador de sistemas tipo Unix, y dicha seguridad comienza por no abusar de los privilegios de *root*.

TEMA 2: INSTALACIÓN BÁSICA

Objetivos

- Comprender los conceptos básicos previos al proceso de instalación del sistema.
- Describir paso a paso los mecanismos a seguir para llevar a cabo la instalación de un sistema operativo Linux.
- Capacitar en los procedimientos óptimos luego de la instalación del sistema.

Contenido

1. Conceptos necesarios previos a la instalación
 - 1.1. Concepto de sistema de archivos
 - 1.2. Concepto de *montaje* de sistema de archivos
 - 1.3. Concepto de *live CD*
 - 1.4. Concepto de partición de disco
 - 1.5. Concepto de espacio de *intercambio*
 - 1.6. Concepto de *MBR (Master Boot Record)*
 - 1.7. Concepto de *gestor de arranque*
 - 1.8. Conceptos relacionados al software
2. Tareas de preparación para la instalación de Linux
 - 2.1. Visión general de la instalación
 - 2.2. Repartición del disco o discos duros
 - 2.3. Requerimientos de la partición de Linux
 - 2.4. Creación del espacio de *intercambio*
 - 2.5. Creación del sistema de archivos
 - 2.6. Instalar el software
 - 2.7. Instalar el gestor de arranque *GRUB*
3. Procedimientos posteriores a la instalación
 - 3.1. Creación de una cuenta de usuario no *root*
 - 3.2. Pedirle ayuda a nuestro sistema
 - 3.3. Edición del archivo */etc/fstab*
 - 3.4. Cerrar el sistema

Bibliografía

Básica

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.

Complementaria

- Roger Baig Viñas, Francesc Aulí Llinàs, “Sistema operativo GNU/Linux básico Primera Edición”, UOC Formación de Posgrado, Software libre, noviembre 2003.
http://www.nodo50.org/cursos/manual_gnu_linux.pdf
- Sebastián Sánchez Prieto, “Sistemas operativos, textos universitarios, segunda edición”, Editorial universidad de Alcalá.

Uno de los compromisos más importantes que como buenos administradores debemos cumplir, es el compromiso de seleccionar una distribución de Linux para su posterior instalación, pero siempre todos nos hacemos la misma pregunta ¿Cuál elegir? La respuesta esta en que de todo ese amplio abanico de distribuciones que Linux pone a nuestra disposición debemos seleccionar la que mejor se ajuste a nuestras necesidades. Algo que no es una tarea fácil, pero que debemos de tomarnos nuestro tiempo para elegir la mejor opción.

1. Conceptos necesarios previos a la instalación

En esta sección vamos a estudiar algunos conceptos que son necesarios antes de introducirnos de lleno en el proceso de la instalación del sistema. A lo largo de todo el tema haremos uso de dichos conceptos. Es recomendable manejar y comprender cada uno de estos conceptos los cuales nos facilitarán la comprensión de un proceso tan delicado e interesante dentro de los sistemas Linux, como es el proceso de instalación.

1.1. Concepto de sistema de archivos

Los sistemas de archivos son elementos fundamentales en los sistemas operativos ya que nos permiten almacenar la información de un modo permanente, así como también estructurar dicha información almacenada.

El sistema operativo Linux es un poco especial puesto que es capaz de trabajar con sistemas de archivos diferentes. Para ello Linux utiliza una capa de abstracción denominada *Virtual File System (VFS)*. Esta capa proporciona un mecanismo de acceso uniforme a cualquier sistema de archivos montado en el sistema.

Dentro de los sistemas de archivos más utilizados actualmente y que son soportados por Linux tenemos:

- **ext2:** Fue el sistema de archivos por defecto para múltiples distribuciones de Linux hasta ser reemplazado por su sucesor *ext3*. La principal desventaja que presenta es que no hace uso de *journaling*, que básicamente aporta estabilidad y mejora el rendimiento de los sistemas de archivos tras una caída inesperada empleando un cuaderno de bitácora o *log* que permite abortar operaciones en caso de que se produzca algún error.
- **ext3:** Es un sistema de archivos que si hace uso de *journaling*, permite actualizar de *ext2* a *ext3* sin perder los datos almacenados ni formatear el disco. Utiliza un árbol binario balanceado e incorpora el asignador de bloques de disco *orlov*, con el fin de mejorar el rendimiento prestado por los anteriores sistemas de archivos *ext2*.
- **FAT32:** Es un sistema de archivos desarrollado para MS-DOS. Cuando se borran y se escriben nuevos archivos tiende a dejar fragmentos dispersos de éstos por todo el soporte. Con el tiempo, esto hace que el proceso de lectura o escritura sea cada vez más lento. La denominada desfragmentación es la solución a este problema. No hace uso de *journaling*. Carece de asignaciones de permisos a los archivos. Linux es capaz de leer y escribir en este tipo de sistema de archivos, mediante la correcta configuración de los parámetros establecidos en el archivo */etc/fstab*.
- **NTFS:** Es un sistema de archivos diseñado específicamente para Windows NT y posteriores, con el objetivo de crear un sistema de archivos eficiente robusto y con seguridad incorporada desde su base. Permite compresión nativa de ficheros, cifrado e incluso *journaling*, pero sólo a partir de Windows Vista. Microsoft no ha liberado su código, pero gracias a técnicas de ingeniería inversa Linux tiene soporte de escritura y lectura sobre este tipo de sistemas de archivos a través del controlador *NTFS-3G*.

- **ReiserFS:** Es un sistema de archivos de propósito general, diseñado e implementado por un equipo de la empresa *Namesys*. Previene el riesgo de corrupción del sistema de archivos mediante la utilización de *journaling*, permite aumentar el tamaño del sistema de archivos mientras este se encuentra montado o desmontado e implementa un esquema para reducir la fragmentación interna llamado *Tail packing*.
- **XFS:** Es un sistema de archivos de 64 bits con *journaling* de alto rendimiento creado por *SGI* (antiguamente, *Silicon Graphics Inc.*). Soporta un sistema de archivos de hasta 9 exabytes (2^{60} bytes), aunque esto puede variar dependiendo de los límites impuestos por el sistema operativo. Estos sistemas de archivos están particionados internamente en grupos de asignación, que no son más que regiones lineares de igual tamaño dentro del sistema de archivos, la idea es que cada grupo gestiona sus *inodos* y su espacio libre de forma independiente.
- **JFS:** Es un sistema de archivos con respaldo de transacciones desarrollado por *IBM*. Diseñado con la idea de conseguir servidores de alto rendimiento y servidores de archivos de altas prestaciones. Al ser un sistema de archivos de 64 bits, *JFS* soporta ficheros grandes y particiones *LFS* (*Large File Support*), lo cual es una gran ventaja para los entornos de servidores.

Estos son algunos de los muchos sistemas de archivos soportados por Linux. Una de las tareas del administrador del sistema antes de llevar a cabo el proceso de instalación es determinar que sistema de archivos es el que mejor se acopla a sus necesidades. Un elemento que juega a nuestro favor es que Linux nos da soporte para diversos sistemas de archivos evitándonos de esta manera limitaciones a solo unos cuantos de ellos.

1.2. Concepto de *montaje* de sistema de archivos

Es muy común tener en una misma máquina uno o varios discos físicos, cada uno de ellos con distintas particiones. En cada una de estas particiones podemos tener un sistema de archivos diferente y es entonces cuando surge la necesidad de poder acceder a archivos ubicados en particiones de cualquiera de los discos. Algunos sistemas operativos establecen la diferenciación representando cada sistema de archivos con una letra diferente. En el caso de Linux se opta por otra alternativa que consiste en ubicar cada sistema de archivos en un directorio diferente, por todos los sistemas de archivos colgando de una única *raíz*. Es por esto que al proceso de colgar un sistema de archivos de un directorio se le conoce como montaje de un sistema de archivos.

Los sistemas de archivos deben ser montados siempre en directorios vacíos denominados puntos de montajes, los cuales deben existir con anterioridad ya que el simple hecho de montar un sistema de archivos no crea el directorio de punto de montaje.

1.3. Concepto de *live CD*

Un *live CD*, también conocido más genéricamente como *liveDistro*, consiste en una distribución de Linux que contiene el núcleo del sistema operativo, normalmente acompañado de un conjunto de aplicaciones, almacenado en un medio extraíble, tradicionalmente un CD, aunque las distribuciones modernas son tan completas que se incluyen en un DVD. La principal característica que presenta el *live CD* es la de ejecutar y testear la distribución contenida dentro del CD sin necesidad de instalar absolutamente nada en el propio disco duro de la máquina, para lo cual se utiliza la memoria RAM como disco duro virtual y el propio CD como sistema de ficheros.

Algunos *live CD* incluyen una herramienta que permite instalar la distribución, que estos llevan en su interior, en el propio disco duro de la máquina. Otra característica que presentan es que mientras no se instale nada no se efectúa ningún cambio en la máquina utilizada.

Para usar un *live CD* es necesario obtener uno, hoy en día la gran mayoría de distribuciones Linux utilizan este mecanismo para su instalación, por lo cual son relativamente fáciles de conseguir por Internet (<http://distrowatch.com>). Lo siguiente que debemos hacer para usar el *live CD* es configurar nuestra máquina para que arranque desde la unidad lectora, reiniciando luego la máquina con el disco en la lectora, con lo que lograremos que el *live CD* se inicie automáticamente.

1.4. Concepto de partición de disco

Desde la introducción del disco duro en los ordenadores IBM PC y compatibles ha existido la posibilidad de dividirlo en particiones. Una partición de disco la forman un conjunto de bloques de disco situados de forma contigua y que pueden ser tratados como un disco lógico independiente, separado del resto. Las razones para emplear varias particiones son las siguientes: Aislar los datos, incrementar la eficiencia del disco y evitar la utilización del disco sin control. El aislamiento de los datos permite garantizar que aunque una partición del disco se corrompa, el resto no se vea afectada. La eficiencia se ve mejorada porque podemos emplear en cada partición distinto tamaño del bloque de datos acorde con las necesidades, todo ello con objeto de encontrar un equilibrio entre fragmentación interna y rapidez de transferencias. Por último, si las áreas de datos las separamos de las áreas de almacenamiento empleadas por el sistema, en distintas particiones, evitaremos que un crecimiento desmesurado de esa área de datos afecte al propio sistema.

Independientemente del sistema de archivos que contenga una partición, existen tres tipos diferentes de particiones:

1. **Partición primaria:** Son las divisiones crudas o primarias del disco, solo pueden haber cuatro de éstas. Depende de una tabla de particiones. Un disco físico completamente formateado, consiste en realidad de una partición primaria que ocupa todo el espacio del disco, y posee un sistema de archivos.
2. **Partición extendida:** Es un tipo de partición que actúa como una partición primaria; sirve para contener infinidad de unidades lógicas en su interior. Fue ideada para romper la limitación de cuatro particiones primarias en un solo disco físico. Solo sirve para contener particiones lógicas. Por lo tanto, es el único tipo de partición que no soporta un sistema de archivos directamente.
3. **Partición lógica:** Ocupa un trozo de partición extendida o la totalidad de la misma, la cual se ha formateado con un tipo específico de sistema de archivos (FAT32, NTFS, ext2, etc.).

1.5. Concepto de espacio de *intercambio*

El espacio de intercambio o comúnmente conocido como *swap* es una zona de intercambio entre la memoria RAM y el disco duro de la máquina. Sirve cuando el sistema operativo tiene toda la memoria RAM ocupada y los programas en ejecución piden más. Es en este momento cuando se empieza a utilizar el espacio de intercambio para guardar zonas de RAM que no se están utilizando, intercambiándolas para que las aplicaciones no se queden sin memoria disponible.

1.6. Concepto de MBR (Master Boot Record)

El *MBR* es el primer sector físico de un dispositivo de almacenamiento de datos, como por ejemplo, un disco duro. Es también conocido como sector cero o sector de arranque principal (*bootsector*).

En la práctica, el *MBR* casi siempre se refiere al sector de arranque el cual está constituido de 512 bytes y estos a su vez están divididos en tres partes (ver figura 2.1):

- **Los primeros 446 bytes:** Contienen el código de arranque. Si la *BIOS* inicia el proceso de arranque desde ese disco, se ejecutará este programa.
- **Los siguientes 64 bytes:** Contienen la tabla de particiones. Esta tabla consta de cuatro entradas, cada una de las cuales mantiene información sobre una partición. Esta es la razón por la que el número de particiones primarias está limitado a cuatro. Cada entrada contiene un descriptor de partición el cual define el tipo de sistema de archivos almacenado en la propia partición (ext2, ext3, FAT32, NTFS, etc.), almacena también información sobre la ubicación de la partición en el disco y un flag que indica si la partición es activa o no. La partición activa es aquella partición en donde el *BIOS* buscará primero el sistema operativo en un disco duro.
- **Los 2 últimos bytes:** Contienen el número mágico 0xAA55 que identifica a este sector como un sector de arranque.



Figura 2.1: Estructura del *MBR*

1.7. Concepto de gestor de arranque

El gestor de arranque es un pequeño programa que se ubica en el disco duro de la máquina para que en el proceso de arranque de la misma podamos elegir qué sistema operativo de los que tenemos instalados queremos arrancar. Aunque el único sistema operativo instalado en nuestra máquina sea Linux, necesitaremos tener instalado un gestor de arranque ya que el sistema de arranque de dicho sistema operativo lo necesita.

Existen varias aplicaciones para realizar este proceso. Las más usuales son *Lilo* (**L**inux **L**oader) y *GRUB* (**G**Rand **U**nified **B**ootloader). Lo único que hacen estas aplicaciones es iniciar el proceso de carga y ejecución del núcleo del sistema operativo que se les haya especificado en su configuración.

Normalmente el gestor de arranque se suele poner en el *MBR* del disco maestro del primer canal IDE o SCSI, que es el primer sitio que la *BIOS* del ordenador inspecciona buscando un programa de estas características.

1.8. Conceptos relacionados al software

Algunos conceptos importantes relacionados con el software son:

- **Paquete:** Es uno o varios programas, librerías o componentes de software empaquetados en un solo archivo preparado para que sea instalado e integrado en el sistema operativo.

En estos paquetes se suelen incluir los ejecutables del programa y sus dependencias y conflictos con otras aplicaciones.

- **Dependencias:** Indican, al instalar un paquete, si necesitan otros programas para que la aplicación funcione correctamente.
- **Conflictos:** Informan de incompatibilidades entre programas instalados y el que queremos instalar.
- **Sistema de gestión de paquetes:** Es aquel que proporciona las herramientas necesarias para poder instalar y gestionar adecuadamente cualquier paquete. Los sistemas de gestión de paquetes están diseñados de tal forma que permitan facilitar la instalación de las nuevas aplicaciones.

Más adelante en este tema entraremos en más detalle sobre cada uno de los conceptos vistos anteriormente. Mientras tanto estos nos serán de gran utilidad para entender un poco mejor cada una de las etapas de la instalación del sistema.

2. Tareas de preparación para la instalación de Linux

Tras haber seleccionado la distribución de Linux que mejor se ajuste a nuestras necesidades, estaremos listos para preparar nuestro sistema para instalar dicha distribución. El proceso de instalación requiere de una planificación, especialmente si en el sistema que vamos a instalar nuestro Linux, ya se está ejecutando otro u otros sistemas operativos. Es por esto que en las siguientes secciones se explicará cómo planificar la instalación de Linux.

2.1. Visión general de la instalación

Aunque cada lanzamiento de Linux es diferente, en general el método utilizado para instalarlo es el mismo:

- **Repartición del disco o discos duros:** Si tenemos otros sistemas operativos ya instalados en nuestro sistema, necesitaremos realizar de nuevo la partición de los discos para poder asignar espacio para nuestro Linux. En la gran mayoría de distribuciones actuales este paso ya se encuentra integrado en el procedimiento de instalación.
- **Iniciación del medio de instalación de Linux:** Cada una de las distribuciones de Linux cuentan con algún tipo de medio de instalación. Generalmente, las distribuciones actuales cuentan con un CD-ROM desde el que se puede realizar el arranque. Al abrir el medio, se abrirá algún tipo de programa de instalación, que nos guiará a través de ella o nos permitirá instalar el software manualmente.
- **Creación de particiones de Linux:** Tras volver a realizar la partición para asignar el espacio para Linux, debemos crear la partición de Linux en el espacio vacío con el programa *fdisk* o con otro programa específico que se cargue durante el proceso de instalación.
- **Creación de sistemas de archivos y espacio de intercambio:** En este momento, crearemos uno o más sistemas de archivos, utilizados para guardar archivos, en las particiones recién creadas. Asimismo, si prevemos utilizar un espacio de intercambio (algo que deberíamos hacer, a no ser que realmente dispongamos de una gran cantidad de memoria física o RAM), crearemos el espacio de intercambio en una de nuestras particiones.

- **Instalar nuestro Linux en el nuevo sistema de archivos:** Por último, debemos instalar el sistema operativo Linux en nuestro sistema de archivos recién creado. Posteriormente, si todo va bien, el resto es sencillo.

Algunos usuarios que cambian entre distintos sistemas operativos a veces se preguntan qué sistema operativo tienen que instalar primero: ¿Linux o el otro sistema? Existen muchos casos de usuarios que han tenido problemas instalando un sistema operativo Windows tras Linux. Casi siempre los sistemas operativos Windows suelen destruir, sin consentimiento alguno, la información del arranque cuando se instalan, por lo que es más recomendable instalarlo de primero y dejar siempre de último la instalación de Linux.

Muchas distribuciones de Linux proporcionan un programa de instalación que nos conducirá a través del todo el proceso y automatizará uno o más pasos previos.

Cuando nos estemos preparando para instalar Linux, el mejor consejo que podemos acatar es tomar notas durante todo el procedimiento de instalación. Debemos escribir todo lo que hagamos, los comandos que escribamos y lo que veamos que nos puede parecer extraño. La idea que se persigue es simple: si se produce algún problema, tendremos que volver atrás en nuestros pasos y descubrir dónde está el error. Instalar Linux no es difícil pero hay muchos detalles que debemos recordar. Necesitaremos un registro de todos estos detalles para poder experimentar con otros métodos si algo nos sale mal. Asimismo, conservar las notas de nuestras experiencias de instalación de Linux es útil cuando deseamos pedir ayuda a alguien.

2.2. Repartición del disco o discos duros

En general, los discos duros se dividen en particiones con una o más de ellas dedicadas a un sistema operativo. Por ejemplo, en un disco duro podemos tener diversas particiones independientes, una dedicada a, por ejemplo Windows, otra a Knoppix y otra a Ubuntu. Si ya tenemos otro sistema operativo instalado en nuestro sistema, tendremos que redimensionar dichas particiones para liberar espacio para almacenar nuestro nuevo sistema operativo Linux. Posteriormente tendremos que crear, dentro de ese mismo espacio libre, una o más particiones Linux para guardar el sistema operativo y el espacio de intercambio de Linux. Este proceso se le denomina repartición.

Muchos sistemas Windows utilizan una sola partición para todo el disco. Para Windows, esta partición generalmente se le suele asignar la letra C:. Si tenemos más de una partición, Windows las denominará D:, E:, y así sucesivamente. En cierto sentido, cada partición actúa como un disco duro independiente.

En el primer sector del disco se encuentra un registro maestro de arranque junto con una tabla de particiones. El registro de arranque, tal como su nombre lo indica, se utiliza para iniciar el sistema. La tabla de particiones contiene información sobre las ubicaciones y tamaños de las particiones.

Existen tres tipos de particiones: primaria, extendida y lógica. De éstas, las particiones primarias son las que se suelen utilizar con mayor frecuencia. Sin embargo, debido al límite del tamaño de la tabla de particiones, podemos tener sólo cuatro particiones primarias en determinados discos, algo que se debe al mal diseño de MS-DOS y Windows; incluso otros sistemas operativos que se originaron en la misma época no tienen este tipo de limitaciones.

La forma de evitar el límite de las cuatro particiones es utilizar una partición extendida. Esta partición, por sí sola, no contiene ningún dato sino que actúa como *contenedor* para particiones lógicas. De tal manera que una partición extendida puede contener hasta cuatro particiones

lógicas, a su vez, una partición lógica puede definirse como partición extendida, la cual puede también contener hasta cuatro particiones lógicas y así sucesivamente.

2.3. Requerimientos de la partición de Linux

Antes de entrar en detalles de cómo se crea la repartición de los discos, necesitamos saber cuánto espacio tendremos que asignar a nuestro Linux.

En los sistemas Unix, los archivos se guardan en un sistema de archivos, básicamente una sección del disco duro formateado para contener archivos. Cada sistema de archivos se asocia a una parte específica del árbol de directorios; por ejemplo, en muchos sistemas existe un sistema de archivos para todos los archivos en el directorio */usr*, otro en el directorio */tmp*, etc. El sistema de archivos *raíz (/)* es el sistema de archivos primario, que se corresponde con el directorio más alto.

En los sistemas Unix, cada sistema de archivos reside en una partición independiente del disco duro. Por ejemplo, si tenemos un sistema de archivos para */* y otro para */usr*, necesitaremos dos particiones para que contengan estos dos sistemas de archivos. Cabe destacar que esto es sólo aplicable a sistemas de archivos, no a directorios. Evidentemente, podemos tener cualquier cantidad de árboles de directorios en el directorio *raíz* dentro del mismo sistema de archivos.

Antes de instalar Linux tendremos que preparar los sistemas de archivos para guardar el sistema operativo. Tenemos que tener al menos un sistema de archivos (el sistema de archivos *raíz*) y, por consiguiente, una partición asignada a Linux. Muchos usuarios de Linux optan por guardar sus archivos en el sistema de archivos *raíz* que, normalmente, es más fácil para administrar diversos sistemas de archivos y particiones. Sin embargo, si se desea, podemos crear múltiples sistemas de archivos para Linux, por ejemplo, podemos utilizar sistemas de archivos independientes para */usr* y para */home*.

¿Por qué es necesario utilizar más de un sistema de archivos? La razón establecida con más frecuencia es la seguridad; si por alguna razón se daña un sistema de archivos, los otros (normalmente) no tendrán daños. Por el contrario, si guardamos todos nuestros archivos en el sistema de archivos *raíz* y, por alguna razón, este sistema de archivos se daña, podemos perder todos nuestros archivos de una sola vez. Sin embargo, es algo que se produce con poca frecuencia; si realizamos copias de seguridad del sistema con regularidad estaremos bastante asegurados.

Por otro lado, el uso de diversos sistemas de archivos tiene la ventaja de poder actualizar fácilmente nuestro sistema sin poner en peligro nuestros preciados datos. Podemos tener una partición para los directorios principales (*home*) de los usuarios y, al actualizar el sistema, dejar esta partición y borrar el resto para poder reinstalar Linux desde el principio. Evidentemente, las distribuciones actuales tienen procedimientos de actualización bastante elaborados pero, de vez en cuando, puede que necesitemos un nuevo arranque. Otra razón para tener múltiples sistemas de archivos es repartir el almacenamiento entre múltiples discos duros. Supongamos que tenemos 300MB libres en un disco duro y 2GB libres en otro; podemos crear un sistema de archivos *raíz* de 300 MB en el primer disco duro y un sistema de archivos */usr* de 2GB en el otro. Es posible tener un solo sistema de archivos que ocupe múltiples unidades utilizando una herramienta denominada *administrador de volumen lógico (LVM, Logical Volume Manager)*, pero realizar esta configuración requiere un considerable conocimiento sobre el tema, a no ser que el programa de instalación de nuestra distribución lo haga de forma automática.

En resumen, Linux requiere, al menos de una partición para el sistema de archivos *raíz*. Si deseamos crear múltiples sistemas de archivos necesitaremos una partición independiente para cada sistema de archivos adicional. Algunas distribuciones de Linux crean automáticamente

particiones y sistemas de archivos, por lo que no tendremos que preocuparnos mucho de este problema.

Otro tema a considerar durante la planificación de nuestras particiones es el espacio de intercambio. El espacio de intercambio es una parte del disco duro utilizada por el sistema operativo cuando existen más procesos activos de los que se pueden mantener en la memoria. Esta situación puede producirse porque los procesos residentes soliciten memoria dinámicamente y no exista suficiente para todos. En estas circunstancias, es necesario buscar un proceso poco activo, y moverlo al espacio de intercambio (el disco duro) con objeto de liberar espacio en la memoria principal para cargar otros procesos. Mientras no haga falta, el proceso extraído de memoria puede quedarse en el disco, ya que ahí no gasta memoria física. Cuando sea necesario, se volverá a hacer un intercambio, pasándolo del disco a memoria RAM.

Una pregunta que casi todos nos hemos hecho a la hora de instalar nuestro sistema operativo Linux es la siguiente ¿en una máquina que cuente con mucha memoria RAM es necesario crearle un espacio de intercambio? Aunque puede funcionar bien sin que le creamos ningún espacio de intercambio, es muy recomendable crearla. La razón es que siempre es bueno quitar de la memoria los procesos poco usados, ya que eso nos permitirá usar la RAM para otras tareas. Por ejemplo: Supongamos que abrimos en un programa de edición de imágenes, como por ejemplo *Gimp*, una imagen muy grande, que nos consume el 80% de la memoria RAM, y que después sin cerrar el programa nos ponemos a hacer varias búsquedas de archivos por nuestro disco duro. Si no podemos llevar al disco ese proceso grande, quiere decir que ha de mantenerse en memoria física; por tanto, las búsquedas sólo tendrán menos del 20% de la memoria RAM y por ende estas pueden ser poco eficientes. Con la creación del espacio de intercambio, se podría llevar a disco el proceso grande, o al menos una parte de él, hacer las búsquedas usando mucha más RAM y luego restaurar el proceso si hace falta.

Existen dos opciones para crear el espacio de intercambio. La primera es utilizar un archivo de intercambio existente desde uno de los sistemas de archivos de Linux. El archivo de intercambio puede ser creado utilizando el siguiente comando:

```
$> mkswap <NombreArchivo>
```

Cabe destacar que existe un problema al hacer uso del comando *mkswap*, el problema es que el *<NombreArchivo>* que le pasemos como parámetro tiene que cumplir un requisito especial: el *<NombreArchivo>* no ha de tener agujeros. Es decir que los bytes del archivo han de estar realmente en el disco. Para solucionar este problema podemos hacer uso del siguiente comando:

```
$> dd if=/dev/zero of=<NombreArchivo> bs=1024 count 65536
```

Mediante este comando estamos creando un archivo llamado *<NombreArchivo>*, que no tendrá agujeros y que su tamaño será de 1024 bloques cada uno de 65536 bytes que suman un total de 64MB. Una vez creado el archivo de intercambio es necesario activarlo para que el sistema haga uso de él, para esto tenemos que hacer uso del siguiente comando:

```
$> swapon <NombreArchivo>
```

En el caso de que quisiéramos desactivar el archivo de intercambio tendremos que hacer uso del siguiente comando:

```
$> swapoff <NombreArchivo>
```

Por último, debemos añadir, en el archivo `/etc/fstab`, el archivo de intercambio que acabamos de crear para que este se inicie junto con el sistema. Una vez hecho esto podemos conocer todos los dispositivos y archivos de intercambio activos en el sistema haciendo uso del siguiente comando:

```
$> cat /proc/swaps
```

La segunda opción es crear una partición de intercambio, la cual es una partición individual para su uso sólo como espacio de intercambio. En la mayoría de los casos está es la opción más utilizada, debido a la facilidad de su implementación la cual ha sido automatizada en la mayoría de procesos de instalación de las distintas distribuciones de Linux.

Otra pregunta que casi todos nos hacemos a la hora de crear nuestro espacio de intercambio es la siguiente ¿qué tamaño le debemos asignar al espacio de intercambio? Hay una regla muy conocida que dice que *el espacio de intercambio ha de ser el doble de la memoria RAM instalada*, pero esta regla hoy en día ya no es válida. Esta regla funcionaba bien antes, cuando siempre se disponía de menos RAM de la que realmente se necesitaba. Tener tres veces más memoria que la física iba bien para la mayoría de los usuarios. Pero en un ordenador moderno que tenga 2GB de RAM, no será necesario gastar 4GB en una partición de intercambio, porque probablemente no se usarán todos los 4GB y por tanto estaremos desperdiciando parte de nuestro disco duro. Hoy en día, la regla habitual usada para decidir el tamaño designado para el espacio de intercambio es *pensar cuánto quisiéramos tener de RAM y cuánto tenemos, y poner como espacio de intercambio la diferencia*. Por ejemplo, si necesitaríamos abrir ficheros de hasta 700 Mb, pero sólo tuviéramos 256Mb de RAM, entonces lo que nos falta (aproximadamente 500Mb) es lo que hemos de poner como espacio de intercambio.

Una vez que hemos visto los requerimientos necesarios en cuanto a particiones se refiere estamos listos para crear una partición. Para ello vamos a hacer uso del comando `fdisk`. Por ejemplo, si deseamos ejecutar `fdisk` en el primer disco duro IDE de nuestro sistema, debemos utilizar el siguiente comando:

```
$> fdisk /dev/hda  
Orden (m para obtener ayuda):
```


Luego, *fdisk* estará esperando que le especifiquemos una opción; podemos utilizar la opción *m* para obtener una lista con todas las opciones que podemos usar con *fdisk*:

\$> fdisk /dev/hda

Orden (m para obtener ayuda): **m**

Command	action
<i>a</i>	Conmuta el indicador de iniciable.
<i>b</i>	Modifica la etiqueta de disco <i>bsd</i> .
<i>c</i>	Conmuta el indicador de compatibilidad con <i>DOS</i> .
<i>d</i>	Suprime una partición.
<i>l</i>	Lista los tipos de particiones conocidos.
<i>m</i>	Imprime este menú.
<i>n</i>	Añade una nueva partición.
<i>o</i>	Crea una nueva tabla de particiones <i>DOS</i> vacía.
<i>p</i>	Imprime la tabla de particiones.
<i>q</i>	Salte sin guardar los cambios.
<i>s</i>	Crea una nueva etiqueta de disco <i>Sun</i> .
<i>t</i>	Cambia el identificador de sistema de una partición.
<i>u</i>	Cambia las unidades de visualización/entrada.
<i>v</i>	Verifica la tabla de particiones.
<i>w</i>	Escribe la tabla en el disco y sale.
<i>X</i>	Funciones adicionales (sólo para usuarios avanzados).

Orden (m para obtener ayuda):

La opción *n* se utiliza para crear una nueva partición. Del resto de opciones normalmente no tendremos que preocuparnos. Para salir de *fdisk* sin guardar ningún cambio, podemos utilizar la orden *q*. Para salir de *fdisk* y escribir todos los cambios realizados en la tabla de particiones del disco, podemos utilizar la opción *w*. Siempre que salgamos con la opción *q* sin escribir nada, podemos curiosear con *fdisk* sin arriesgarnos a dañar nuestros datos. Sólo cuando hagamos uso de la opción *w* podremos causar un desastre potencial sobre nuestros datos, siempre y cuando hagamos algo mal.

Lo primero que debemos hacer es ver nuestra tabla de particiones actual y escribir en un lugar seguro dicha información para consultarla posteriormente. Podemos utilizar la opción *p* para ver esta información. Es recomendable copiar la información en una libreta tras realizar cualquier cambio en la tabla de particiones. Si por alguna razón nuestra tabla de particiones se dañase, no podríamos acceder a ningún dato de nuestro disco duro, aunque los propios datos sigan estando ahí. Pero si utilizamos nuestras notas, podremos restablecer la tabla de particiones ejecutando de nuevo *fdisk* y eliminando y recreando las particiones con los parámetros escritos previamente en nuestra libreta. No nos debemos olvidar de guardar la tabla de particiones establecida cuando hayamos terminado.

Éste es un ejemplo de una tabla de particiones, de un disco duro muy pequeño, donde los bloques, los sectores y los cilindros son las unidades organizativas del disco duro:

Orden (m para obtener ayuda): **p**

Disco /dev/hda: 16 cabezas, 38 sectores, 683 cilindros

Unidades = cilindros de 608 * 512 bytes

Dispositivo	Inicio	Comienzo	Fin	Bloques	Id	Sistema
/dev/hda1 *	1	1	203	61693	6	DOS 16-bit >=32M

Orden (m para obtener ayuda):

En este ejemplo tenemos una sola partición de Windows en `/dev/hda1`, compuesta por 61693 bloques (unos 60Mb). (Salvo que se establezca de forma diferente, un bloque en Linux por defecto suele ocupar 1024 bytes.) Esta partición se inicia en el cilindro número 1 y finaliza en el cilindro número 203. En este disco, tenemos un total de 683 cilindros, por lo que quedan 480 cilindros en los que se pueden crear particiones.

Para crear una partición, podemos utilizar la opción *n*. En este ejemplo vamos a crear dos particiones primarias, `/dev/hda2` y `/dev/hda3`, para Linux:

```
Orden (m para obtener ayuda): n
Acción de la orden
e Partición extendida
p Partición primaria (1-4)
p
```

Aquí, *fdisk* nos está preguntando qué tipo de partición es la que deseamos crear: extendida (*e*, *extended*) o primaria (*p*, *primary*). En nuestro ejemplo, estamos creando sólo particiones primarias por lo que debemos escribir una *p*.

```
Números de partición (1-4):
```

Aquí, *fdisk* nos está solicitando que escribamos el número que le queremos asignar a nuestra nueva partición; como la partición 1 ya se está utilizando, el número de nuestra primera partición de Linux será el 2.

```
Números de partición (1-4): 2
Primer cilindro (204-683):
```

Ahora *fdisk* nos está solicitando el número del cilindro inicial de la partición. Como están sin utilizar los cilindros desde el 204 hasta el 683, vamos a utilizar el primer cilindro disponible (el 204). No hay ninguna razón para dejar un espacio vacío entre las distintas particiones:

```
Primer cilindro (204-683): 204
Último tamaño + tamaño o + tamañoM o + tamañoK (204-683):
```

Ahora *fdisk* nos está pidiendo el tamaño de la partición que deseamos crear. Podemos especificar un número de cilindro de finalización o un tamaño en bytes, kilobytes o megabytes. Como queremos que nuestra partición tenga un tamaño de MB, vamos a especificar `+80M`. Cuando especificamos de esta forma el tamaño de una partición *fdisk* redondea el tamaño de partición real al siguiente número de cilindros:

```
Último tamaño + tamaño o + tamañoM o + tamañoK (204-683): +80M
```

Ahora estamos preparados para crear nuestra segunda partición de Linux. Para este ejemplo vamos a crear una partición con un tamaño de 10MB:

Orden (m para obtener ayuda): **n**
 Acción de la orden
e Partición extendida
p Partición primaria (1-4)
p
 Números de partición (1-4): **3**
 Primer cilindro (474-683): **474**
 Último tamaño + tamaño *o* + tamaño *M* o + tamaño *K* (474-683): **+10M**

Al final miramos la tabla de particiones con la opción *p*, para cerciorarnos de que todo quedo bien configurado. Una vez más es mejor escribir toda esta información, especialmente los tamaños de los bloques de las nuevas particiones. Tendremos que conocer los tamaños de las particiones para poder crear los sistemas de archivos. Asimismo, tendremos que comprobar que ninguna de nuestras particiones se superpone:

Orden (m para obtener ayuda): **p**

Disco /dev/hda: 16 cabezas, 38 sectores, 683 cilindros
 Unidades = cilindros de 608 * 512 bytes

Dispositivo	Inicio	Comienzo	Fin	Bloques	Id	Sistema
/dev/hda1 *	1	1	203	61693	6	DOS 16-bit >=32M
/dev/hda2	204	204	473	82080	83	Linux nativo
/dev/hda3	474	474	507	10336	83	Linux nativo

Como podemos ver, /dev/hda2 es ahora una partición con un tamaño de 82080 bloques, los cuales corresponden aproximadamente con 80MB. Y /dev/hda3 tiene ahora 10336 bloques, los cuales corresponden con aproximadamente 10MB.

Cabe destacar que la mayoría de distribuciones Linux requieren el uso de la opción *t* con el comando *fdisk*, para establecer que la partición de intercambio se le va a asignar el tipo: *intercambio Linux*, la cual corresponde con el código número 82. Podemos imprimir la lista de códigos de tipos de particiones conocidas mediante el uso de la opción *l* y utilizar posteriormente la opción *t* para establecer el tipo de partición de intercambio que corresponde con el tipo: *intercambio de Linux*. De ese modo, el software de instalación podrá encontrar automáticamente las particiones de intercambio basándose en el tipo.

En el ejemplo anterior, los cilindros restantes del disco, numerados desde el 508 hasta el 683 quedarán sin utilizar. Podemos dejar espacio sin utilizar en el disco por si deseamos crear posteriormente particiones adicionales, en caso de estar seguros de no querer crear particiones posteriormente, no es recomendable dejar nunca espacio sin utilizar ya que estaremos desperdiciando el espacio existente dentro de nuestro disco duro.

Por último, podemos utilizar la opción *w* para escribir los cambios en el disco y salir de *fdisk*:

Orden (m para obtener ayuda): **w**
\$>

Cabe destacar que ninguna de las opciones establecidas durante la ejecución de *fdisk* tienen efecto hasta que no se proporcione la opción *w*, por lo que podemos jugar con distintas

configuraciones y guardarlas únicamente cuando estemos completamente seguros de que esas son las particiones que queremos asignar a nuestro sistema. De lo contrario debemos usar la opción *q* para salir de *fdisk* en cualquier momento sin guardar los cambios.

Algunas distribuciones de Linux requieren el reinicio del sistema tras la ejecución de *fdisk* para permitir que los cambios en la tabla de particiones surtan efecto antes de instalar el sistema. Las versiones más modernas de *fdisk* actualizan automáticamente la información de las nuevas particiones en el núcleo, por lo que no es necesario un reinicio. Sin embargo, es recomendable reiniciar el sistema tras la modificación de las particiones dentro del disco duro.

2.4. Creación del espacio de *intercambio*

Si decidimos utilizar una partición de intercambio para la RAM virtual, ahora ya estamos preparados para hacerlo. Muchas distribuciones requieren de la creación y activación de un espacio de intercambio antes de instalar el sistema operativo. Si disponemos sólo de una pequeña cantidad de RAM física, el procedimiento de instalación puede fallar, a no ser que hayamos habilitado alguna cantidad de espacio de intercambio.

El comando utilizado para preparar una partición de intercambio, como vimos anteriormente, es *mkswap*, pero ahora adopta la forma siguiente:

```
$> sudo mkswap -c <partition>
```

Siendo *partition* el dispositivo que hace referencia a la partición de intercambio. Por ejemplo, si el dispositivo que hace referencia a nuestra partición de intercambio es */dev/hda3*, entonces el comando quedaría de la siguiente manera:

```
$> sudo mkswap -c /dev/hda3
```

La opción *-c* le indica a *mkswap* que busque bloques defectuosos en la partición cuando cree el espacio de intercambio. Los bloques defectuosos son puntos en el medio magnético que no contienen datos correctamente, algo que se produce raramente en los discos duros actuales. Es recomendable hacer siempre uso de la opción *-c* para que *mkswap* revise nuestro disco en busca de bloques defectuosos. Esta opción los excluirá automáticamente para que no se puedan utilizar.

Tras formatear el espacio de intercambio, tendremos que habilitarlo para su uso por el sistema. Normalmente, el sistema habilita automáticamente el espacio de intercambio en el momento del arranque. Sin embargo, como todavía no hemos instalado el sistema operativo, tenemos que habilitarlo nosotros mismos manualmente. El comando para habilitar el espacio de intercambio, tal y como lo vimos anteriormente, es *swapon* y adopta la siguiente forma:

```
$> swapon <partition>
```

Siguiendo con el ejemplo anterior para habilitar el nuevo espacio de intercambio creado en */dev/hda3* debemos utilizar el siguiente comando:

```
$> swapon /dev/hda3
```

2.5. Creación del sistema de archivos

Antes de poder utilizar las particiones de Linux para guardar nuestros archivos, primero tenemos que crear el sistema de archivos en ellas. La creación de un sistema de archivos es parecida al formateo de una partición en Windows o en otros sistemas operativos. Existen diversos tipos de sistemas de archivos disponibles para Linux. Cada tipo de sistema de archivos tiene su propio formato y conjunto de características, como la longitud de los nombres de los archivos, el tamaño máximo del archivo, etc. Linux admite además diversos tipos de archivos de terceros, como el sistema de archivos de Windows (FAT32 y NTFS).

Los tipos de sistemas de archivos utilizados con más frecuencia por los sistemas operativos Linux son los sistemas de archivos *ext2* (*Second Extended Filesystem*) y *ext3* (*Third Extended Filesystem*). Los sistemas de archivos *ext2* y *ext3* son dos de los sistemas de archivos más eficaces y flexibles; admiten nombres de archivos de hasta 256 caracteres y tamaños del sistema de archivos de hasta 32 terabytes. Al momento de elegir entre que tipo de sistema de archivos utilizar es recomendable el uso de *ext3*, ya que este presenta una importante ventaja sobre *ext2*, esta ventaja es el uso de cuaderno de bitácora o *log* (*journaling*), que en pocas palabras permite crear un sistema de archivos tolerante a fallos en el cual la integridad de los datos está asegurada porque las modificaciones de la meta-información de los ficheros son primero grabadas en un registro cronológico antes que los bloques originales sean modificados.

Para crear un sistema de archivos *ext3*, podemos utilizar el comando siguiente:

```
$> mke2fs -j -c /dev/hda2
```

Cabe destacar que para crear un sistema de archivos *ext2* también se utiliza el comando *mke2fs*; pero la opción *-j* es la que hace posible la creación de un sistema de archivos *ext3*.

2.6. Instalar el software

Finalmente ya estamos preparados para instalar el software en nuestro sistema. Cada distribución tiene un mecanismo diferente para hacerlo. Muchas distribuciones disponen de un programa automático que nos conduce a través de la instalación. En otras distribuciones tendremos que montar los sistemas de archivos en un determinado subdirectorio, por ejemplo */mnt*, y copiar el software manualmente. Las distribuciones actuales en CD-ROM suelen presentar la opción de instalar una parte del software en el disco duro y dejar la mayoría del mismo en el CD-ROM. Normalmente este procedimiento se denomina *sistema de archivos activo*. Este tipo de sistemas de archivos activos son muy cómodos para probar Linux antes de comprometernos definitivamente a instalarlo en el disco duro.

Algunas distribuciones ofrecen distintos métodos para instalar el software. Por ejemplo, podemos instalar el software directamente desde una partición de Windows en el disco duro en lugar de hacerlo desde disquetes o podemos instalarlo sobre una red TCP/IP vía *ftp* o *NFS*.

Por ejemplo, con la distribución *Slackware* debemos seguir el siguiente procedimiento:

1. Crear particiones con *fdisk*.
2. Opcionalmente, crear un espacio de intercambio con *mkswap* y habilitarla mediante *swapon*.
3. Ejecutar el programa *setup* para instalar el software. Luego es el programa el que nos conduce a través de un sistema de menús.

En contraste con el mecanismo de instalación de software de la distribución del ejemplo anterior, al final de este capítulo analizaremos un caso práctico, en el cual se lleva a cabo de inicio a fin una instalación de una distribución de Linux conocida como Ubuntu, en esta observaremos que todos los pasos previos y posteriores a la instalación, tanto del software como del propio sistema, son muy automatizados e intuitivos, algo que permite que usuarios inexpertos en el mundo de Linux se puedan introducir a él desde el momento en que se inicia el proceso de instalación hasta que deciden que jamás van a dejar de usarlo. Este factor de automatización e intuitividad en el proceso de instalación es algo que muchas distribuciones Linux, como Debian o RedHat, no han podido conseguir y por ende añaden a la gran mayoría de usuarios inexpertos un mayor grado de complejidad, tanto que se deciden por no instalar el sistema.

Las distribuciones modernas de Linux pueden contener unos mil o más paquetes en diversos CD-ROM, pero básicamente existen tres métodos para seleccionar el paquete de software:

- **Selección por tarea:** Éste es el medio más fácil de selección para los principiantes. No hay que pensar si es necesario un determinado paquete. Simplemente debemos escoger si un determinado equipo Linux va a actuar como estación de trabajo, como máquina de desarrollo o como enrutador (*router*) de red y el programa de instalación elegirá los paquetes apropiados. Normalmente podremos redefinir la selección manualmente o volver atrás al programa de instalación posteriormente.
- **Selección de paquetes individuales por serie:** Con este mecanismo de selección, todos los paquetes se agrupan en series como *Redes*, *Desarrollo* o *Imágenes*. Aquí podremos seleccionar los paquetes individuales. Este método requiere más decisiones que si elegimos realizar una selección por tarea ya que tenemos que elegir si necesitamos cada uno de los paquetes. Sin embargo, podemos omitir una serie completa de paquetes si estamos seguros de nos estar interesados en las funciones que ofrecen.
- **Selección de paquetes individuales ordenados alfabéticamente:** Este método sólo es útil si ya sabemos qué paquetes deseamos instalar; en caso contrario, será bastante complicado.

Elegir un método de selección no excluye el uso del resto. La mayoría de distribuciones ofrecen dos o más de los mecanismos de selección mencionados anteriormente.

A pesar de todo puede que siga siendo difícil elegir un paquete. Las buenas distribuciones ofrecen una breve descripción de cada paquete en pantalla para facilitar la selección del paquete correcto pero, en caso de duda es mejor no elegirlo, ya que siempre podemos volver atrás y añadir paquetes posteriormente.

Las distribuciones modernas tienen una opción sensacional denominada registro de dependencia. Algunos paquetes sólo funcionan cuando están instalados otros paquetes, por ejemplo un visor gráfico puede necesitar bibliotecas gráficas especiales para importar archivos. Con el registro de dependencia, la instalación del programa puede informar sobre dichas dependencias y nos permitirá seleccionar automáticamente el paquete deseado junto con todos los paquetes de los que depende. A no ser que estemos seguros de lo que estamos haciendo, es mejor aceptar siempre las ofertas de cumplimiento de dependencias automáticas para evitar que el paquete que necesitamos instalar no funcione adecuadamente más adelante. Los programas de instalación pueden ayudarnos de distintas formas a seleccionar y evitar errores. Por ejemplo, el programa de instalación puede negarse a iniciar la instalación de un determinado paquete si anulamos la selección de un paquete que es absolutamente crucial para que se inicie incluso el sistema más mínimo, como la estructura básica de directorios, o puede buscar una mutua exclusión, como cuando sólo podemos tener un paquete u otro pero no ambos.

2.7. Instalar el gestor de arranque *GRUB*

Cada distribución proporciona distintos medios para poder arrancar el nuevo sistema Linux una vez instalado todo el software. Muchas veces, el procedimiento de instalación nos sugiere la creación de un disquete de arranque, que contiene un núcleo de Linux configurado para utilizar el sistema de archivos *raíz* recién creado. Para poder iniciar Linux debemos arrancarlo desde este disquete; el control se transfiere al disco duro tras el arranque. En otras distribuciones, este disquete de arranque es el propio disquete de instalación. Si el sistema no contiene una disquetera, algo muy común en muchos sistemas modernos, debemos asegurarnos de utilizar otros medios para arrancar Linux, como el inicio directo desde un CD-ROM. Muchas distribuciones nos proporcionan la opción de instalar *GRUB* en el disco duro. *GRUB* es un gestor de arranque múltiple que reside en el registro maestro de arranque y se usa comúnmente para iniciar dos o más sistemas operativos instalados en un mismo ordenador. Además nos permite seleccionar, mediante una lista, cuales de todos los sistemas operativos queremos iniciar en el momento del arranque.

Para instalar *GRUB* con éxito, necesitamos conocer mucha información sobre la configuración de nuestro disco duro: por ejemplo, cuáles son las particiones que contienen los distintos sistemas operativos, cómo iniciar cada sistema operativo, etc. Al instalar *GRUB*, muchas distribuciones intentan *adivinar* los parámetros apropiados para nuestra configuración. A veces, la instalación *GRUB* automatizada proporcionada por algunas distribuciones puede fallar y dejar destrozado el disco de arranque maestro, no obstante, es poco probable que se produzca algún daño en los datos reales de nuestro disco duro.

Además de *GRUB* existen otros gestores de arranque, incluyendo el más antiguo de Linux, Lilo (*Linux LOader*).

3. Procedimientos posteriores a la instalación

Tras completar la instalación del sistema operativo Linux, deberíamos ser capaces de reiniciar el sistema, iniciar una sesión como *root* y empezar el mismo. Cada distribución tiene un método diferente de hacerlo.

Sin embargo, antes de seguir adelante por nuestra cuenta, hay algunas tareas que debemos llevar a cabo para evitar problemas posteriores.

3.1. Creación de una cuenta de usuario no *root*

Para poder empezar a utilizar el sistema es necesario crear una cuenta de usuario.

¿Por qué todo esto? Todos los sistemas Linux tienen cuentas preinstaladas, como *root*. Sin embargo, la cuenta *root* esta diseñada exclusivamente para objetivos administrativos. Como *root*, tendremos todo tipo de privilegios y podremos acceder a todos los archivos de nuestro sistema. Sin embargo, utilizar la cuenta *root* puede ser peligroso, especialmente cuando estamos empezando a descubrir el nuevo mundo de Linux. Como no existen restricciones sobre lo que se puede hacer al iniciar la sesión con esta cuenta, es muy fácil escribir mal un comando, eliminar archivos accidentalmente, dañar nuestro sistema de archivos, etc. Debemos iniciar sesión como *root* sólo cuando tengamos que ejecutar tareas de administración del sistema, cómo editar archivos de configuración, instalar un nuevo software, etc.

Para un uso normal, debemos crear una cuenta de usuario estándar. Los sistemas Linux tienen una seguridad integrada que evita que los usuarios estándar eliminen archivos de otros usuarios y dañen recursos importantes, como los archivos de configuración del sistema. Como

usuario estándar, nos estaremos protegiendo a nosotros mismos ante nuestros propios errores, especialmente si no tenemos experiencia en la administración de un sistema Linux.

Muchas distribuciones de Linux proporcionan herramientas para crear nuevas cuentas. Estos programas normalmente utilizan comandos como *useradd* o *adduser* para llevar a cabo la creación de nuevos usuarios. Como *root*, al llamar a uno de estos comandos, aparecerá un resumen de uso del mismo y podremos crear una nueva cuenta de forma sencilla.

Las distribuciones más modernas proporcionan una herramienta de administración del sistema genérica para diversas tareas, una de las cuales es la creación de una nueva cuenta de usuario.

3.2. Pedirle ayuda a nuestro sistema

Los sistemas Linux proporcionan su ayuda en formas de páginas de manual. Estas páginas de manual nos proporcionan ayuda sobre determinados comandos o archivos de configuración que necesitamos utilizar pero que no sabemos como hacerlo, su uso es sumamente importante ya que son una buena herramienta que nos resuelve los problemas cuando estamos en apuros. Para obtener ayuda de las páginas del manual debemos hacer uso del comando *man*, por ejemplo para obtener información sobre el comando *adduser* debemos usar el siguiente comando:

```
$> man adduser
```

En algunas distribuciones las páginas del manual se proporcionan como un paquete opcional, por lo que no estarán disponibles a no ser que optemos por instalarlas. Sin embargo, es muy aconsejable que lo hagamos. Muchas veces nos sentiremos perdidos sin ellas.

Asimismo, puede que falten o estén incompletas determinadas páginas del manual de nuestro sistema. Dependerá de lo completa que sea la distribución que hayamos elegido y de cómo estén actualizadas las páginas del manual.

Las páginas del manual de Linux también documentan llamadas del sistema, funciones de bibliotecas, formatos de archivos de configuración y elementos básicos del núcleo.

Muchas distribuciones también proporcionan documentación en formato HTML que se puede leer con cualquier explorador web, como Firefox, Konqueror, Opera, etc. Por último hay archivos de documentación con texto común que se pueden leer con cualquier editor de texto o simplemente con el comando *less* o *more*.

Si no podemos encontrar documentación para un determinado comando, también podemos probar a ejecutarlo con la opción *-h* o *-help*. La mayoría de comandos proporcionan un breve resumen de su uso si se escriben con esta opción.

3.3. Edición del archivo */etc/fstab*

Para estar seguros de que todos los sistemas de archivos de Linux estarán disponibles al reiniciar el sistema, tendremos que editar el archivo */etc/fstab*, que se encarga de describir nuestros sistemas de archivos. Muchas distribuciones generan automáticamente el archivo */etc/fstab* durante la instalación. Sin embargo, si tenemos sistemas de archivos adicionales que no hayan sido utilizados durante el proceso de instalación, puede que necesitemos agregarlos al archivo */etc/fstab* para que estén disponibles. En este archivo también deben de incluirse las particiones de intercambio.

Para poder acceder a un sistema de archivos, éste tiene que estar montado en el sistema. El montaje de un sistema de archivos asocia dicho sistema de archivos a un determinado directorio. Por ejemplo, el sistema de archivos *raíz* se monta en `/`, el sistema de archivos de usuario se monta en `/usr`, y así sucesivamente.

El sistema de archivos *raíz* se monta automáticamente en `/` al iniciar Linux. Sin embargo, el resto de sistemas de archivos se tienen que montar de forma individual. Normalmente es una tarea que se consigue con el siguiente comando:

```
$> mount -av
```

Este comando montará cualquier sistema de archivos que aparezca listado en el archivo `/etc/fstab`. Por consiguiente, para que nuestros sistemas de archivos se monten automáticamente en el momento del arranque, tendremos que incluirlos en el archivo `/etc/fstab`. Evidentemente siempre podremos montar los sistemas de archivos manualmente utilizando el comando `mount` tras el inicio del sistema, pero es una tarea innecesaria.

Éste es un ejemplo del archivo `/etc/fstab` resumido ya que hemos omitido los dos últimos parámetros en cada línea por ser opcionales y no ser importantes para nuestro análisis. En este ejemplo, el sistema de archivos raíz se encuentra en `/dev/hda1`, el sistema de archivos `/home` se encuentra en `/dev/hdb2` y la partición de intercambio se encuentra en `/dev/hdb1`.

```
# /etc/fstab
# device directory type options
#
/dev/hda1    /           ext3    defaults
/dev/hdb2    /home      ext3    defaults
/dev/hdb1    none       swap    sw
```

Las líneas que empiezan con el carácter `#` son comentarios. Como podemos observar, el archivo `/etc/fstab` está compuesto de una serie de líneas. El primer campo de cada línea es el nombre del dispositivo de la partición, como `/dev/hda1`. El segundo campo es el punto de montaje, es decir, el directorio donde se monta el sistema de archivos. El tercer campo es el tipo. El cuarto campo es para las opciones de montaje. Si utilizamos este ejemplo como modelo, debemos ser capaces de agregar entradas para cualquier sistema de archivos que no aparezca listado en el archivo `/etc/fstab`.

Después de añadir las entradas para los sistemas de archivos adicionales que no se están cargando automáticamente al iniciar el ordenador, debemos ejecutar el siguiente comando:

```
$> mount -a
```

También, podemos reiniciar el sistema para que los cambios tengan efecto.

3.4. Cerrar el sistema

Nunca debemos reiniciar nuestro sistema Linux pulsando el conmutador de restablecimiento o apagarlo directamente mediante el botón de apagado. Igual que sucede con la mayoría de sistemas Unix, Linux copia escrituras de disco en la memoria cache. Por consiguiente, si reiniciamos de repente el sistema sin haberlo cerrado limpiamente, podemos dañar los datos que se encuentren en nuestros discos duros. Sin embargo, debemos tener en cuenta que presionar las combinaciones de teclas `Ctrl+Alt+Supr` simultáneamente es generalmente seguro: el núcleo atrapa la secuencia de teclas y se las pasa al proceso `init` que, a su vez, inicia un cierre limpio del sistema.

Sin embargo, la configuración de nuestro sistema podría tener reservadas las combinaciones de teclas *Ctrl+Alt+Supr* para el administrador del sistema, para que de esta forma los usuarios normales no puedan cerrar el servidor de red del que depende todo un departamento. Para establecer permisos para utilizar esta combinación de teclas, debemos crear un archivo denominado `/etc/shutdown.allow` con una lista de nombres de todos los usuarios a los que se les permitirá cerrar la máquina mediante esta combinación de teclas.

El método más sencillo de cerrar el sistema es utilizar el comando *shutdown*. Como ejemplo, para cerrar y reiniciar el sistema inmediatamente, podemos utilizar el siguiente comando:

```
$> sudo shutdown -r now
```

Así lograremos cerrar y reiniciar limpiamente nuestro sistema.

La mayoría de distribuciones también proporcionan el comando *halt*, el cual llama inmediatamente a *shutdown*. Existen otras distribuciones que proporcionan el comando *poweroff*, que en realidad cierra el equipo y lo apaga. Y otras que proporcionan el comando *reboot*, que en realidad cierra y reinicia el equipo limpiamente.

TEMA 3: INSTALACIÓN, ACTUALIZACIÓN Y COMPILACIÓN DE PROGRAMAS

Objetivos

- Determinar con precisión el porque y el cuando realizar actualizaciones de programas en los sistemas Linux.
- Desarrollar capacidades de manipulación sobre los mecanismos disponibles para la instalación, desinstalación y actualización de software sobre sistemas Linux.
- Estudiar los sistemas de gestión de paquetes más importantes y más utilizados en la mayoría de distribuciones Linux.
- Describir los pasos a seguir para instalar manual y seguramente un software sin ayuda de los sistemas de gestión de paquetes.
- Analizar la funcionalidad que cumplen las bibliotecas estáticas y dinámicas existentes dentro del sistema y conocer cómo estas son utilizadas por las distintas aplicaciones.

Contenido

1. Actualización de software
2. Procedimiento general de actualización
3. Sistema de gestión de paquetes *RPM*
4. Sistema de gestión de paquetes *.deb*
 - 4.1. Utilizar *dpkg*
 - 4.2. Utilizar *apt*
 - 4.2.1. El corazón de *apt*
 - 4.2.2. Tipos de paquetes según su prioridad
 - 4.2.3. Grado de dependencia entre paquetes
 - 4.2.4. Acciones sobre paquetes
 - 4.2.5. Estado de instalación de los paquetes
 - 4.2.6. Utilizar *apt-cache*
 - 4.2.7. Utilizar *apt-get*
5. Software no proporcionado en paquetes
 - 5.1. Actualización de bibliotecas

Bibliografía

Básica

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada “, Editorial Anaya Multimedia, 2006.

Complementaria

- Roger Baig Viñas, Francesc Aulí Llinàs, “Sistema operativo GNU/Linux básico Primera Edición”, UOC Formación de Posgrado, Software libre, noviembre 2003.
http://www.nodo50.org/cursos/manual_gnu_linux.pdf

En este tema vamos a aprender a actualizar el software de nuestro sistema. Aunque la mayoría de distribuciones de Linux proporcionan algún medio automatizado para instalar, eliminar y actualizar paquetes de software específicos en el sistema, a veces es necesario instalar algún software manualmente. Además es recomendable conocer cuales son los procedimientos adoptados por el sistema para llevar a cabo tanto instalaciones, eliminaciones y actualizaciones de software.

Generalmente, es mucho más fácil instalar, desinstalar y actualizar cualquier software utilizando los sistemas de gestión de paquetes que actualmente son proporcionados en la gran mayoría de distribuciones Linux. Ya que si no utilizamos un sistema de gestión de paquetes, las instalaciones, eliminaciones y actualizaciones pueden ser más complicadas que en la mayoría de sistemas operativos comerciales. Aunque se encuentren disponibles binarios compilados previamente, puede que tengamos que descomprimirlos y desempaquetarlos desde un almacén de archivos. También puede que tengamos que crear vínculos simbólicos o establecer variables de entorno para que los binarios sepan dónde buscar los recursos que utilizan. En otros casos, tendremos que compilar el software personalmente desde las fuentes, una tarea que la verdad no es nada sencilla ni agradable.

1. Actualización de software

Actualmente Linux se mueve con mucha rapidez. Debido a la naturaleza cooperativa del proyecto, siempre se encuentra disponible un nuevo software y los programas siempre se están actualizando con versiones más modernas. Con este constante desarrollo, ¿cómo se puede esperar estar al día en las versiones más recientes de un sistema de software? La respuesta es que en algunos casos nunca estaremos al día en las versiones. En esta sección vamos a analizar porque y cuando debemos realizar una actualización y además vamos a actualizar diversos elementos importantes de nuestro sistema.

¿Cuándo debemos realizar una actualización? En general, debemos considerar actualizar una parte de nuestro sistema sólo cuando tengamos una necesidad demostrada para hacerlo. Por ejemplo, si sabemos que existen nuevas versiones de algunas aplicaciones que solucionan fallos importantes, es decir, fallos que realmente afectan al uso personal de la aplicación, entonces podemos considerar la actualización de dicha aplicación. Si la nueva versión del programa proporciona nuevas opciones que nos pueden resultar útiles o mejora el rendimiento con respecto a la versión que actualmente tenemos, también es recomendable realizar la actualización. Cuando nuestra máquina está conectada a Internet, otra buena razón para actualizar sería solucionar una brecha de seguridad sobre la que nos hayamos informado recientemente. Sin embargo, actualizar sólo por tener la versión más moderna de un programa determinado no es nada recomendable. En algunos casos muy extraños las versiones más modernas son incluso regresiones, es decir, introducen errores o fallos de rendimiento en comparación con una versión anterior.

A veces el proceso de actualización puede ser una tarea difícil. Por ejemplo, puede que necesitemos actualizar a un programa que requiere las versiones más modernas del compilador, bibliotecas y otro software para poderlo ejecutar. La actualización de este programa también requerirá la actualización de otras partes del sistema, un proceso que nos puede consumir mucho tiempo. Por otro lado, esta consideración podría ser un argumento para mantener la gran mayoría del software existente en nuestro sistema siempre actualizado: si nuestro compilador y bibliotecas son actuales, actualizar el programa en cuestión no será ningún problema.

Ahora que ya sabemos porque debemos realizar una actualización, se nos presenta un nuevo problema: ¿cuál es el mejor método de actualización? Algunos creen que es mucho más fácil actualizar completamente el sistema volviendo a instalar todo desde el principio siempre que se lance una nueva versión de nuestra distribución favorita. De esta forma, no tendremos que preocuparnos de las diversas versiones de software que trabajan juntas.

Sin embargo, la reinstalación no es un buen plan de actualización. La mayoría de distribuciones actuales de Linux no están diseñadas para actualizarse de esta forma y una completa reinstalación puede ser muy compleja o consumir mucho tiempo. Asimismo, si prevemos realizar una actualización de esta manera, generalmente perderemos todas las modificaciones y personalizaciones realizadas sobre nuestro sistema y tendremos que realizar copias de seguridad de los directorios personales de los usuarios y de cualquier otro archivo importante que se pudiera eliminar, o al menos comprometer, durante una reinstalación. Por último, adoptar una solución drástica para realizar una actualización significa que, en la práctica, probablemente tendremos que esperar más que si actualizamos el software cuando se anunciarán fallos de seguridad importantes. En realidad, no hay muchos cambios de una versión a otra, por lo que una completa reinstalación normalmente no es necesaria y se puede evitar con un poco de conocimiento sobre las actualizaciones disponibles.

2. Procedimiento general de actualización

Tal y como lo hemos analizado en las secciones anteriores, normalmente es más fácil y mejor actualizar sólo las aplicaciones que tengamos que actualizar. Por ejemplo, si nunca utilizamos el editor de texto *Emacs* en nuestro sistema, ¿por qué preocuparnos de mantenerlo actualizado con la versión más reciente? Seguramente no necesitamos estar completamente al día con aplicaciones que no utilizamos con frecuencia.

La regla general que todo administrador de sistemas debe tener siempre en mentes es que si existe algún software que no es tan indispensable para el correcto funcionamiento del sistema y que si no se actualiza tampoco lo pone en riesgo, entonces dicho software no necesita ser actualizado.

Los sistemas Linux modernos proporcionan diversos métodos de actualización de software, algunos de estos métodos son manuales y son considerados los más flexibles pero presentan la desventaja de ser difíciles de llevar a cabo y complicados de actualizar. Sin embargo, existen otros métodos que son bastante automatizados. En esta sección vamos a examinar tres métodos diferentes para llevar a cabo las tareas relacionadas con el software de nuestro sistema: el primer método consiste en utilizar el sistema de paquetes *RPM (Red Hat Package Manager)*, el segundo en utilizar el sistema de paquetes Debian (*APT, Advanced Packaging Tool* y *dpkg, Debian PacKaGe*) y el tercero consiste en el método manual.

Es importante resaltar que utilizar paquetes y sistemas de gestión de paquetes es muy cómodo y que incluso usuarios con poca experiencia, pueden realizar estas técnicas ya que ahorran mucho tiempo e implican menos conocimientos. A continuación se presentan algunas de las ventajas que nos ofrece la utilización de paquetes y sistemas de gestión de paquetes:

- Todo lo que pertenece a un paquete de software se encuentra en un solo archivo descargable.
- Podemos eliminar un paquete de software completamente del sistema sin que peligren otros paquetes.
- Los sistemas de gestión de paquetes mantienen una base de datos de dependencias y, por tanto, pueden registrar las dependencias asociadas a un determinado paquete antes de que este sea instalado. Por ejemplo, pueden indicarnos si tenemos que instalar una versión más moderna de una biblioteca para ejecutar una determinada aplicación que estemos a punto de instalar. Y rechazará la eliminación de un paquete de biblioteca siempre que existan paquetes instalados que utilicen las bibliotecas que proporciona dicho paquete.

Evidentemente, los sistemas de gestión de paquetes también tienen algunos inconvenientes. Un problema genérico que presentan es que cuando utilizamos el sistema de gestión de paquetes, tenemos que instalarlo todo a través de los paquetes. En caso contrario, no podremos registrar las dependencias. Por la misma razón, no es recomendable mezclar sistemas de gestión de paquetes diferentes.

Es muy probable que todos los días se actualice algún programa que estemos utilizando, lamentablemente esto sucede con mucha frecuencia debido a los fallos de seguridad. Algunos administradores de sistemas meticulosos insisten en revisar con regularidad los informes de seguridad y en actualizar todos los paquetes manualmente utilizando los medios explicados en las siguientes secciones para poder controlar todos los aspectos del sistema y asegurarse de que ningún cambio afecta a la funcionalidad existente. Ésta es una noble causa a la que nos podemos dedicar y además es factible en sistemas con propósitos delicados y con un conjunto de software limitado; como por ejemplo, servidores de correo, servidores web, servidores de bases de datos, sistemas de tiempo real, etc. No obstante, para sistemas de propósitos más general, mantener todo lo que utilizamos actualizado con regularidad se convierte en una tarea muy importante. Por esta razón, la gran mayoría de distribuciones actuales nos proporcionan servicios de actualización automatizados.

3. Sistema de gestión de paquetes *RPM*

RPM, que originalmente se aplicó al administrador de paquetes de la distribución *Red Hat* pero que ahora es un nombre por derecho propio, es una herramienta que automatiza la instalación de los binarios de software y recuerda cuáles son los archivos necesarios (dependencias) para asegurarse de que el software se va a ejecutar correctamente. A pesar del nombre, *RPM* no es específico de *Red Hat* ya que se utiliza en muchas otras distribuciones, incluyendo entre las principales a: *SuSE*, *Fedora* y *Mandriva*. Al utilizar el sistema de gestión de paquetes *RPM* suministrado en dichas distribuciones, la instalación, actualización y desinstalación del software se convierte en una tarea mucha más fácil y sencilla.

La idea básica de *RPM* es que existe una base de datos de paquetes y archivos que pertenecen a un paquete. Al instalar un nuevo paquete, la información sobre éste se registra en la base de datos. A continuación, si deseamos desinstalar el paquete para cada archivo del mismo, *RPM* comprueba si otros paquetes instalados están utilizando también ese archivo. Si éste es el caso, el archivo no se elimina.

Asimismo, *RPM* registra las dependencias. Cada paquete puede depender de uno o más paquetes. Al instalar un paquete, *RPM* comprueba si los paquetes de los que depende el nuevo paquete se encuentran ya instalados. En caso contrario, nos informa sobre las dependencias incumplidas y rechaza la instalación del paquete.

Las verificaciones de dependencias entre paquetes también se utilizan en el proceso de eliminación de los mismos: cuando deseamos desinstalar un paquete del que dependen todavía otros paquetes, *RPM* también nos lo hacer saber y rechaza la ejecución de dicha tarea.

Sin embargo, la incrementada comodidad del uso de paquetes *RPM* tiene un precio: en primer lugar, para los desarrolladores de programas, les es significativamente más difícil crear un paquete *RPM* que simplemente empaquetarlo todo en un archivo *tar*, y en segundo lugar, no se puede recuperar sólo un archivo de un paquete *RPM*; hay que instalarlo todo o nada.

Si ya disponemos de un sistema que utiliza el sistema de gestión de paquetes *RPM*, instalar paquetes *RPM* es muy fácil. Supongamos que tenemos un paquete *RPM* denominado *SuperFrob-4.i386.rpm*. Los paquetes *RPM* siempre tienen la extensión *.rpm*; *i386* indica que se trata de un paquete binario compilado para máquinas *Intel x86*. Este paquete puede ser instalado de la siguiente manera:

```
$> sudo rpm -i SuperFrob-4.i386.rpm
```

En lugar de la opción `-i`, también podemos utilizar la versión de nombre largo de esta opción:

```
$> sudo rpm - --install SuperFrob-4.i386.rpm
```

Si todo va bien, no habrá ninguna salida. Si deseamos que *RPM* ofrezca más detalle, podemos utilizar:

```
$> sudo rpm -ivh SuperFrob-4.i386.rpm
```

Este comando imprimirá el nombre del paquete y un número de marcas `#` para que podamos ver el progreso de la instalación.

Si el paquete que deseamos instalar depende de otro paquete que todavía no está instalado, nos aparecerá un mensaje como el siguiente:

```
$> sudo rpm -i SuperFrob-4.i386.rpm
failed dependencies:
    frobnik-2 is needed by SuperFrob-4
```

Si observamos este mensaje, podemos buscar el paquete *frobnik-2* e instalarlo primero. Evidentemente, este paquete puede depender a su vez de otros paquetes.

Si deseamos actualizar un paquete que ya está instalado en el sistema, podemos utilizar la opción `-U` o su versión larga `--update`:

```
$> sudo rpm -U SuperFrob-5.i386.rpm
```

La desinstalación de un paquete se lleva a cabo con la opción `-e` o `--erase`. En este caso no es necesario especificar el nombre del archivo del paquete, sino únicamente el nombre del paquete y su número de versión:

```
$> sudo rpm -e SuperFrob-5
```

Aparte de las opciones descritas anteriormente que alteran el estado del sistema, la opción `-q` proporciona diversos tipos de información sobre todo lo que se ha grabado en la base de datos *RPM*. Con la opción `-q` podemos conseguir realizar las siguientes tareas:

- Buscar el número de versión de un paquete instalado:

```
$> sudo rpm -q SuperFrob
SuperFrob-5
```

- Obtener una lista de todos los paquetes instalados:

```
$> sudo rpm -qa
SuperFrob-5
OmniFrob-3
...
glibc-2.3.4-23.4
```

- Saber a qué paquete pertenece un determinado archivo:

```
$> sudo rpm -qf /usr/bin/dothefrob
SuperFrob-5
$> sudo rpm -qf /home/kalle/.xinitrc
file /home/kalle/.xinitrc is not owned by any package
```

- Mostrar información sobre el paquete especificado:

```
$> sudo rpm -qi rpm
Name: rpm                      Relocations: (not relocatable)
Version: 4.1.1                 Vendor: SUSE LINUX Products GmbH,
Nuernberg, Germany
Release: 208.2   Build Date: 11 Jun 2005 01:53:04
AM CEST
Install date: 28 Jun 2005 10:02:18 AM CEST   Build Host: purcell.suse.de
Group: System/Packages Source   RPM: rpm-4.1.1-208.2.src.rpm
Size: 5970541                    License: GPL
Signature: DSA/SHA1, Sat 11 Jun 2005 01:58:41 AM CEST, Key ID a84edae89c800aca
Packager: http://www.suse.de/feedback
Summary: The RPM Package Manager
Description:
RPM Package Manager is the main tool for managing the software packages of the SuSE Linux
distribution.
...
Distribution: SuSE Linux 9.3 (i586)
```

- Mostrar los archivos que se van a instalar para el paquete especificado:

```
$> sudo rpm -qpl SuperFrob-5.i386.rpm
/usr/bin/dothefrob
/usr/bin/frobhelper
/usr/doc/SuperFrob/Installation
/usr/doc/SuperFrob/README
/usr/man/man1/dothefrob.1
```

Si nos llegamos a enfrentar con la instalación de un paquete *RPM* en una distribución basada en Debian, la cual utiliza paquetes *.deb* y no *.rpm*, las cosas se nos pueden complicar un poco más. Sin embargo, podemos utilizar el comando *alien*, el cual convierte entre diversos formatos de paquetes y se incluye en la mayoría de distribuciones Linux actuales.

Por ejemplo, para pasar de *.rpm* a *.deb* con el comando *alien* podemos ejecutar lo siguiente:

```
$> alien - -to-deb SuperFrob-5.i386.rpm
```

Cabe destacar que para trabajar con paquetes *.rpm* la gran mayoría de distribuciones cuenta con diversas herramientas como son: *apt4rpm*, *up2date* (Red Hat), *urpmi* (Mandriva), *YaST* (SuSE), *YUM* (Fedora y Yellow Dog Linux).

4. Sistemas de gestión de paquetes *.deb*

En esta sección vamos a analizar y aprender a utilizar sistemas de gestión de paquetes *.deb*. Haremos énfasis específicamente en los sistemas *dpkg* y *apt*. Mientras *dpkg* es una interfaz de bajo nivel para el administrador de paquetes de Debian, la mayoría de funciones normalmente se controlan a través de la familia de programas *apt* (ver figura 3.1) o de interfaces como: *dselect*, *aptitude*, *gnome-apt*, *synaptic* o *KPackage*.



Figura 3.1: Esquema de elementos basados en texto de la familia de programas *apt*

4.1. Utilizar *dpkg*

Tras *RPM*, el administrador de paquetes más conocido para las distribuciones de Linux es *dpkg*, que se utiliza para administrar paquetes *.deb*. Como su nombre implica, el formato *.deb* está unido a la distribución Debian, por lo que también se utiliza en distribuciones basadas en Debian, como Ubuntu y Kubuntu, Libranet y Xandros. Al igual que el formato *RPM*, el formato *.deb* también registra las dependencias de los archivos para asegurarse de que el sistema es consistente.

Las diferencias técnicas existentes entre los dos formatos son realmente pequeñas; aunque los formatos *RPM* y *.deb* son incompatibles. Por ejemplo, no se puede instalar directamente un paquete Debian en Red Hat. Sin embargo, podemos utilizar el comando *alien* para convertir los paquetes *.deb* para otras distribuciones (y viceversa). La diferencia principal existente entre los formatos es que los paquetes *.deb* se crean utilizando herramientas que ayudan a asegurarse de tener un esquema consistente y generalmente conforme a determinadas directivas principalmente, el *Debian Policy Manual*, proporcionado en el paquete *debian-policy* que ayudan a los desarrolladores de software a crear paquetes de alta calidad.

Instalar paquetes *.deb*, utilizando *dpkg*, en un sistema Debian es bastante fácil. Por ejemplo, si tenemos un paquete denominado *superfrob_4-1_i386.deb*, podemos instalarlo de la siguiente manera:

```
$> dpkg -i superfrob_4-1_i386.deb
Selecting previously deselected package superfrob.
(Reading database ... 159540 files and directories currently installed.)
Unpacking superfrob (from superfrob_4-1_i386.deb) ...
Setting up superfrob(4-1) ...
```

Si falta alguna dependencia del paquete *superfrob*, *dpkg* nos emitirá un mensaje de aviso:

```
$> dpkg -i superfrob_4-1_i386.deb
Selecting previously deselected package superfrob.
(Reading database ... 159540 files and directories currently installed.)
Unpacking superfrob (from superfrob_4-1_i386.deb) ...
Dpkg: dependency problems prevent configuration of superfrob:
superfrob depends on frobnik (>>2); however:
Package frobnik is not installed.
Dpkg: error processing superfrob (--install):
dependency problems - leaving unconfigured
Errors were encountered while processing:
superfrob
```

El mensaje de salida indica que para poder instalar el paquete *superfrob_4-1_i386.deb* completamente, es necesario tener instalada la versión 2 o posterior de *frobnik*. Los archivos en el paquete están instalados pero puede que no funcionen hasta que esté instalado también el paquete *frobnik*.

Al contrario que *RPM*, *dpkg* no distingue entre la instalación de un nuevo paquete y la actualización de uno existente; en ambos casos se utiliza la opción *-i* (o *--install*). Por ejemplo si queremos actualizar *superfrob* utilizando un paquete *superfrob_5-1_i386.deb* recién descargado, simplemente escribiríamos:

```
$> dpkg -i superfrob_5-1_i386.deb
(Reading database ... 159546 files and directories currently installed.)
Preparing to replace superfrob 4-1 (using superfrob_5-1_i386.deb) ...
Unpacking replacement superfrob ...
Setting up superfrob(5-1) ...
```

Para desinstalar un paquete, podemos utilizar la opción *-r* (*--remove*) o *-P* (*--purge*). La opción *--remove* eliminará la mayoría del paquete, pero conservará cualquier archivo de configuración existente, mientras que *--purge* eliminará también los archivos de configuración del sistema. Por ejemplo, para eliminar totalmente *superfrob* deberíamos utilizar el siguiente comando:

```
$> dpkg -P superfrob
(Reading database ... 159547 files and directories currently installed.)
Removing superfrob ...
```

También podemos utilizar *dpkg* para buscar los paquetes que están instalados en nuestro sistema utilizando la opción *-l* (*--list*):

```
$> dpkg -l
Desired=Unknow/Install/Remove/Purge/Hold
/ Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
// Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:uppercase=bad)
// Name Version Description
+++--=====
ii a2ps 4.13b-15 GNU a2ps 'Anything to PostScript' converted
ii aalib1 1.4p5-10 ascii art library
ii abcde 2.0.3-1 A Better CD Encoder
...
ii zlib1g-dev 1.1.3-19 compression library - development
```

Las tres primeras líneas de la salida están designadas para indicar lo que significan las tres primeras columnas anteriores al nombre de cada paquete. Normalmente se suele mostrar *ii*, para indicar que el paquete está instalado correctamente. Si no es así, debemos escribir *dpkg --audit* para obtener una explicación de lo que está mal en nuestro sistema y cómo podemos hacer para solucionarlo.

También podemos utilizar la opción *-l* con un nombre de paquete; por ejemplo, podríamos conocer cual es la versión de *superfrob* instalada en nuestro sistema utilizando el siguiente comando:

```
$> dpkg -l superfrob
Desired=Unknow/Install/Remove/Purge/Hold
/ Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
// Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:uppercase=bad)
// Name Version Description
+++--=====
ii superfrob 4-1 The superfrobulator
```

dpkg también se puede utilizar para conocer el paquete al que pertenece un determinado archivo:

```
$> dpkg --search /bin/false
shellutils: /bin/false
$> dpkg --search /home/kalle/.xinitrc
dpkg: /home/kalle/.xinitrc not found
```

También podemos ver información sobre un paquete o archivo *.deb* instalado:

```
$> dpkg --status dpkg
Package: dpkg
Essential: yes
Status: install ok installed
...
Original-Maintainer: Dpkg Developers <team@dpkg.org>
```

```
$> dpkg --info reportbug_1.43_all.deb
new debian package, versión 2.0.
size 66008 bytes: control archive= 1893 bytes.
40 bytes, 2 lines conffiles
...
reportbug is designed to be use don systems gith an installed mail transport agent, like eximo r
sendmail; however, you can edit the configuration file and send reports using any availabe mail
```

dpkg también puede listar los archivos y directorios incluidos en un archivo *.deb*:

```
$> dpkg --contents superfrob_4-1_i386.deb
-rwxr-xr-x root/root 44951 2002-02-10 12:16:48 ./usr/bin/dothefrob
-rwxr-xr-x root/root 10262 2002-02-10 12:16:52 ./usr/bin/frobhelper
```

4.2. Utilizar *apt*

Además de *dpkg*, Debian y otras distribuciones basadas en Debian, proporcionan la familia de programas *apt*. Cabe destacar, que algunas distribuciones basadas en *RPM* también incluyen ahora *apt* (*apt4rpm* o *apt-rpm*). *apt* está diseñado como un sistema independiente del archivo que puede controlar múltiples formatos de paquetes. Quizá la opción más importante de *apt* sea su capacidad de resolver automáticamente las dependencias; si, por ejemplo, *superfrob* requiere la versión 2 o posterior de *frobink*, *apt* intentará buscar *frobink* desde las fuentes disponibles (repositorios que incluyen CD-ROM, espejos locales e Internet).

apt pone a nuestra disposición esencialmente dos herramientas: *apt-get* y *apt-cache*. La primera herramienta puede ser utilizada única y exclusivamente por el usuario *root* del sistema, ya que es la herramienta de gestión de paquetes: instalación, desinstalación, actualización, etc., mientras que la segunda, al ser orientada a la búsqueda de información dentro de la base de datos, ya sean paquetes instalados o sin instalar, puede ser utilizada por cualquier usuario.

4.2.1. El corazón de *apt*

Existe un archivo que es el corazón de la configuración del sistema de gestión de paquetes de las distribuciones basadas en Debian. Este archivo generalmente suele estar ubicado en el directorio */etc/apt/* y recibe el nombre de *sources.list*. Se trata de un fichero de texto, como la gran mayoría de ficheros de configuración en los sistemas Linux, el cual puede ser editado manualmente mediante cualquier editor de texto en modo consola, o bien haciendo uso de editores de textos en modo gráfico.

El contenido de este fichero dependerá en gran medida de si contamos con una conexión a Internet o no. Pero en cualquiera de los casos nunca debemos olvidarnos de ejecutar la instrucción siguiente como *root*, una vez que hayamos modificado el fichero:

```
$> sudo apt-get update
```

Si no lo hiciéramos, la base de datos del sistema de paquetes no se actualizaría y, en consecuencia, ninguno de los cambios realizados surgiría efecto.

Cada línea del archivo */etc/apt/sources.list* hace referencia una fuente de paquetes y los campos van separados por un espacio. En primer lugar se especifica si la fuente de paquetes es de paquetes binarios mediante la clausula *deb* o si es de código fuente mediante la clausula *deb-src*. En el segundo campo se especifica la forma de acceder a éstos que puede ser mediante CD-

ROM, http, ftp, etc. seguido de la dirección de acceso. Los campos restantes hacen referencia al tipo de paquetes al que queremos acceder por esta línea.

En los casos en los que dispongamos de un acceso a Internet tendremos la gran ventaja de disponer siempre de las últimas versiones de los paquetes. Además esta forma es considerada como la más cómoda para trabajar con paquetes, ya que no tenemos ni siquiera que preocuparnos de insertar el CD correspondiente para hacer una instalación. De lo que tenemos que cerciorarnos, antes que nada, es que el contenido de `/etc/apt/sources.list` sea el correcto y una vez que hayamos editado y salvado el archivo, ejecutemos el comando: **sudo apt-get update** para asegurarnos de obtener el acceso a los nuevos paquetes.

En el caso de no disponer de una conexión a Internet debemos optar, sin dudarlo, por utilizar el juego de CD de nuestra distribución para conseguir instalar los distintos paquetes que necesitamos. Para ello debemos insertar el CD-ROM en la unidad lectora de CD y utilizar el comando: **sudo apt-cdrom add** para incluir sus contenidos en la base de datos. Una vez que hemos tecleado el comando anterior, debemos repetir el mismo proceso para todos y cada uno de los CD de nuestra distribución. Asimismo, podemos utilizar el mismo procedimiento para incorporar datos procedentes de CD-ROM no oficiales. Una vez que hemos añadido todos los conjuntos de CD, debemos teclear el comando: **sudo apt-get update** para tener disponibles los nuevos paquetes. A medida que necesitemos instalar algún paquete mediante el comando: **sudo apt-get install <nombre_paquete>**, el sistema nos indicará cuál es el CD-ROM que debemos introducir en la unidad lectora de CD para poder instalar dicho paquete.

4.2.2. Tipos de paquetes según su prioridad

Dentro de los sistemas de paquetes de la familia de programas *apt* se distinguen cinco tipos de paquetes distintos clasificados según su grado de dependencia con el mismo sistema. Por orden decreciente de prioridad se clasifican como:

Required: Se trata de paquetes indispensables para el correcto funcionamiento del propio sistema.

Important: Se trata de paquetes que deberían estar presentes en cualquier sistema Linux.

Standard: Se trata de paquetes que comúnmente se encuentran en un sistema Linux. Por lo general son aplicaciones de tamaño reducido, pero que ya no son indispensables para el sistema.

Optional: Se trata de paquetes que pueden estar o no, presentes en un sistema Linux. Entre otros, dentro de este grupo se hallan todos los paquetes referentes al sistema gráfico, ya que éste no se considera indispensable. En realidad, en muchos servidores, con el objeto de aumentar su rendimiento se prescinde del entorno gráfico.

Extra: Son paquetes que, bien presentan conflictos con paquetes con prioridad superior a la suya o bien porque requieren de configuraciones especiales que no los hacen aptos para ser integrados como *optional*.

Podemos determinar a qué grupo pertenece un paquete en concreto mediante el uso del siguiente comando: **apt-cache show <nombre_paquete>** y consultar el contenido del campo *Priority*:

4.2.3. Grado de dependencia entre paquetes

apt se caracteriza por su gran consistencia a la hora de gestionar las dependencias que existen entre paquetes. Puede, por ejemplo, que una determinada aplicación que queremos instalar dependa de una librería y, en consecuencia, de otro paquete que no tengamos instalado.

En este caso, *apt* nos informará de esta dependencia y nos preguntará si queremos que, junto con la aplicación que vamos a instalar, se instale el paquete que contiene dicha librería. Las relaciones entre paquetes se clasifican de la manera siguiente:

depends: El paquete que queremos instalar depende de estos paquetes y, por consiguiente, si queremos que este paquete funcione correctamente, debemos permitir que *apt* instale el resto de ellos.

recommends: El responsable del mantenimiento del paquete ha estimado que normalmente los usuarios que vayan a instalar este paquete también usarán los que él recomienda.

suggests: Son los paquetes que se sugieren para obtener un mayor rendimiento del paquete que vamos a instalar.

conflicts: El paquete que vamos a instalar no funcionará correctamente si estos otros paquetes están presentes en el sistema.

replaces: La instalación de este paquete implica la desinstalación de otros paquetes.

provides: El paquete que vamos a instalar incorpora todo el contenido de los paquetes mencionados.

Podemos determinar las dependencias de un paquete mediante el siguiente comando: **apt-cache depends <nombre_paquete>**.

4.2.4. Acciones sobre paquetes

Los siguientes flags indican las acciones que se pueden llevar a cabo sobre un determinado paquete:

unknown: Nunca se ha hecho referencia a dicho paquete.

install: Se quiere instalar o actualizar el paquete.

remove: Se quiere desinstalar el paquete, pero manteniendo sus ficheros de configuración (comúnmente situados en el directorio */etc*).

purge: Se quiere desinstalar por completo el paquete, inclusive sus ficheros de configuración.

hold: Se quiere indicar que no se quiere realizar ninguna operación sobre este paquete, es decir, que se mantenga hasta nuevo aviso su versión y su configuración.

4.2.5. Estado de instalación de los paquetes

Los distintos estados de instalación de los paquetes son:

installed: El paquete ha sido instalado y configurado correctamente.

not-installed: El paquete no está instalado en el sistema.

unpacked: El paquete ha sido desempaquetado, pero no configurado.

half-installed: El paquete ha sido desempaquetado y se ha iniciado el proceso de configuración, pero por alguna razón éste no ha terminado.

config-files: Sólo existen, en el sistema, los archivos de configuración de dicho paquete.

4.2.6. Utilizar *apt-cache*

Como ya se ha dicho, *apt-cache* es un comando orientado al análisis del sistema de paquetes y, por tanto, al no ser un arma potencialmente peligrosa para el sistema, es accesible a todos sus usuarios. Los parámetros más utilizados para este comando son los siguientes:

search <pattern>: Busca en la base de datos los paquetes cuyo nombre contenga *pattern* o en cuya descripción aparezca *pattern*. Si el resultado es una lista muy extensa debido a que *pattern* es muy general, podemos utilizar *pipes* (*/*) y *grep* para filtrar los resultados.

show <nombre_paquete>: Muestra una informa completa y detallada acerca del paquete especificado como parámetro. Además, muestra las dependencias directas y las reversas del paquete.

policy <nombre_paquete>: Informa acerca del estado de instalación, la versión y revisión del paquete especificado como parámetro, y además muestra su procedencia.

depends <nombre_paquete>: Muestras las dependencias de paquetes asociadas al paquete especificado como parámetro.

4.2.7. Utilizar *apt-get*

apt-get es el comando que se utiliza para gestionar los paquetes del sistema. Por este motivo su uso está restringido al *root* del sistema. Los parámetros más utilizados para este comando son los siguientes:

install <nombre_paquete>: Instala el paquete. Si éste depende de paquetes que no se encuentran en el sistema, *apt* nos informará de ello, y nos preguntará si junto con el paquete en cuestión queremos instalar los paquetes de los que depende y que no están instalados; por lo general es interesante seguir los consejos de *apt*.

update: Actualiza la base de datos de *apt*. Este comando debe ejecutarse cada vez que modificamos el archivo */etc/apt/sources.list*.

upgrade: Fuerza la actualización de todos los paquetes instalados en el sistema a la última versión disponible.

remove <nombre_paquete>: Elimina el paquete, sin eliminar los ficheros de configuración, de cara a posibles reinstalaciones.

remove --purge <nombre_paquete>: Elimina por completo el paquete, incluyendo sus archivos de configuración.

clean: Elimina las copias caducadas de los paquetes que se han ido instalando, proceso en el cual se almacena de manera automática una copia del paquete sin desempaquetar en el directorio */var/cache/apt/archives* cuando se instala un paquete. Este comando es muy útil de cara a liberar espacio del disco duro, ocupado por ficheros que, probablemente, nunca más serán utilizados.

autoclean: Elimina todas las copias no desempaquetadas de los paquetes, independientemente de su vigencia.

Al contrario de la mayoría de los comandos de Linux, las acciones de los comandos *apt* se especifican sin guiones. *apt-get* admite algunas opciones, pero se utilizan sólo para cambiar el comportamiento de la acción principal especificada.

Además de las herramientas de línea de comandos, se han desarrollado interfaces gráficas basadas en texto fáciles de utilizar. Una de las herramientas más avanzadas es *KPackage*, que forma parte del entorno de escritorio *KDE* pero se puede utilizar con otros escritorios, como *GNOME*, aunque este último cuenta con su propia herramienta llamada *Synaptic*.

5. Software no proporcionado en paquetes

Existen muchos software interesantes que se ofrecen fuera de los sistemas de paquetes, aunque a medida que se conocen más, los desarrolladores suelen ofrecerlos en paquetes de Linux. Para poder instalar o actualizar aplicaciones que no existen como paquetes, tendremos que obtener la versión más moderna del software, que normalmente se encuentra disponible como archivos comprimidos *gzip* o como archivos empaquetados con *tar*. Este paquete puede presentarse de diversas formas. La más común es la distribución binaria, en la que los binarios y archivos relacionados se archivan y preparan para desempaquetarlos en nuestro sistema, y las distribuciones de fuentes, en la que se proporciona el código fuente (o partes del mismo) para el software y no tenemos que emitir ningún comando para compilarlo e instalarlo en nuestro sistema. Las bibliotecas compartidas facilitan la distribución del software en forma binaria; siempre que tengamos una versión de las bibliotecas instaladas compatible con la biblioteca auxiliar utilizada para crear el programa, no tendremos ningún problema. Sin embargo, en muchas ocasiones, es mucho más fácil y recomendable instalar una versión del programa como fuentes. De este modo, no sólo está disponible el código fuente para su inspección y posterior desarrollo sino que, además, podemos crear la aplicación específicamente para nuestro sistema, con nuestras propias bibliotecas. Muchos programas nos permiten especificar determinadas opciones en el momento de la compilación, como incluir de forma selectiva diversas opciones en el programa cuando lo instalamos. Este tipo de personalización no es posible si utilizamos binarios configurados previamente.

Existe además un problema de seguridad cuando se instalan los binarios sin el código fuente. Aunque en los sistemas Linux la existencia de virus es casi inexistente, no es difícil escribir un *caballo de troya* (programa que parece hacer algo útil pero que, en realidad, causa daños en el sistema) para ellos. Un virus en su sentido clásico es un programa que ataca al *anfitrión* y se ejecuta cuando se ejecuta éste. En los sistemas Linux, este tipo de acción requiere privilegios *root* para hacer algún tipo de daño y, si los programadores pueden obtener dichos privilegios, probablemente no tengan que preocuparse por un virus. Por ejemplo, alguien podría escribir una aplicación con la opción de eliminar todos los archivos del directorio personal del usuario que ejecute el programa. Como el programa se va a ejecutar con los permisos del usuario que lo ejecute, el propio programa tendrá la capacidad de hacer este tipo de daño. Evidentemente, el mecanismo de seguridad de Linux evita el daño a los archivos de otros usuarios o a cualquier archivo del sistema propiedad de *root*. Aunque tener el código fuente no evita necesariamente que sucedan estos incidentes relacionados con virus (ya que generalmente, nunca leemos todo el código fuente de todos los programas que compilamos en nuestro sistema), al menos, nos proporciona una forma de verificar lo que está haciendo realmente el programa. Asimismo, si el código fuente se encuentra disponible, es muy probable que alguien lo examine con detalle, por lo que utilizar las fuentes es algo más seguro; sin embargo, no podemos contar con ello.

Existen técnicas para verificar los paquetes binarios, principalmente los paquetes firmados. El empaquetador puede firmar un paquete con su clave *PGP* (*Pretty Good Privacy*) y las herramientas del paquete como *RPM* tienen medios para verificar dicha firma. Sin embargo, no nos podemos fiar de que el empaquetador haya empaquetado todo correctamente y sin malas intenciones. La firma indica que el paquete en realidad proviene de quien dice provenir y que no ha sido manipulado en su camino desde el empaquetador hasta nuestro disco duro.

De todos modos, tratar con distribuciones de fuentes y binarias del software es muy simple. Si el paquete se proporciona como archivo *tar*, primero debemos utilizar la opción **tar tvf <nombre_archivo>** para determinar cómo se han guardado los archivos. En el caso de las distribuciones binarias, podremos desempaquetar el archivo *tar* directamente en nuestro sistema (por ejemplo, desde `/usr`). Al hacerlo, debemos asegurarnos de eliminar cualquier versión antigua del programa y sus archivos de apoyo (los que no se sobrescriben con el nuevo archivo *tar*). Si el antiguo ejecutable se encuentra antes que el nuevo en nuestra ruta de acceso, seguirá ejecutándose la antigua versión a no ser que la eliminemos.

Las distribuciones de fuentes son algo más complicadas. Pero tenemos que desempaquetar las fuentes en un directorio propio. La mayoría de sistemas utilizan para esta tarea el directorio `/usr/src`. Como normalmente no es necesario ser *root* para poder montar un paquete de software (aunque normalmente necesitaríamos permisos de *root* para instalar el programa tras su compilación), es recomendable que como administradores de sistemas establezcamos permisos de escritura para todos los usuarios en `/usr/src` con el comando:

```
$> sudo chmod 1777 /usr/src
```

Así permitiremos que cualquier usuario cree subdirectorios en `/usr/src` y coloque ahí archivos. El primer *l* en el modo evita que los usuarios eliminen subdirectorios unos de otros. Ahora podemos crear un subdirectorio en `/usr/src` y desempaquetar el archivo *tar* ahí o podemos desempaquetarlo directamente desde `/usr/src`, si el archivo contiene un subdirectorio propio.

Cuando las fuentes están disponibles, el siguiente paso es leer cualquier archivo README (LÉAME) e INSTALL (INSTALAR) o cualquier nota de instalación incluida en las fuentes. Casi todos los paquetes incluyen este tipo de documentación. El método básico utilizado para montar la mayoría de programas (ver figura 3.2) es el siguiente:

1. Revisar el archivo *Makefile*. Este archivo contiene instrucciones para *make*, que controla el compilador para montar programas. Muchas aplicaciones requieren la edición de aspectos menores del archivo *Makefile* para adaptarse a nuestro propio sistema. Las notas de instalación son las que nos indican si es necesario llevar a cabo dichas ediciones. Si no hay ningún archivo *Makefile* en el paquete, puede que primero tengamos que generarlo (ver punto tres para saber cómo proceder).
2. Posiblemente tengamos que editar otros archivos asociados con el programa. Algunas aplicaciones requieren la edición de un archivo denominado *config.h*; esta tarea la analizaremos en las instrucciones de instalación.
3. Posiblemente tengamos que ejecutar una secuencia de comandos de configuración. Esta secuencia de comandos se utiliza para determinar cuáles son las utilidades disponibles en nuestro sistema, algo necesario para montar algunas aplicaciones más complejas.

Específicamente, cuando las fuentes no contienen un archivo *Makefile* en el directorio de nivel superior, sino un *Makefile.in* y un archivo *configure*, el paquete se ha creado con el sistema *autoconf*. En este caso, algo que es cada día más común, debemos ejecutar una secuencia de comandos como la siguiente:

```
$> ./configure
```

Se debe utilizar `./` para ejecutar el archivo *configure* local y no otro programa *configure* que pudiera encontrarse accidentalmente en nuestra ruta de acceso. Algunos paquetes

nos permiten pasar opciones a *configure* que normalmente habilitan o deshabilitan opciones específicas del paquete; podemos conocer dichas opciones con el comando: `./configure --help`. Una vez que hayamos ejecutado la secuencia de comandos de *configure*, podemos seguir con el siguiente paso.

4. Ejecutar *make*. Normalmente, este comando ejecuta los comandos de compilación apropiados, tal como aparecen en el archivo *Makefile*. En muchas ocasiones, tendremos que proporcionar al comando *make* un *destino*, como *make all* o *make install*. Éstos son dos destinos comunes; el segundo normalmente no es necesario porque se puede utilizar para montar todos los destinos listados en un archivo *Makefile* (por ejemplo, si el paquete incluye diversos programas pero sólo se compila uno de forma predeterminada); el último se utiliza normalmente para instalar los ejecutables y los archivos de apoyo en el sistema tras la compilación. Por esta razón, *make install* normalmente se debe ejecutar como *root*. Incluso tras la instalación, normalmente existe una diferencia importante entre los programas instalados desde las fuentes o desde un paquete binario. Los primeros normalmente se instalan en el directorio `/usr/local` de forma predeterminada, algo muy poco común en el caso de paquetes binarios.

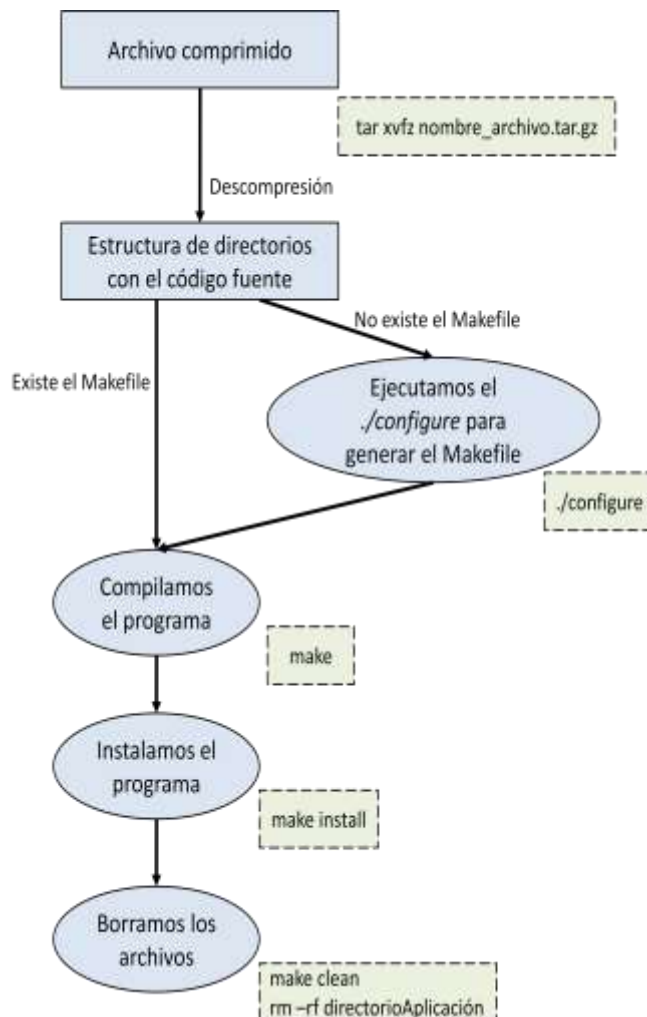


Figura 3.2: Esquema básico utilizado para montar la mayoría de programas

Podemos encontrar problemas al compilar o instalar un nuevo software de nuestro sistema, especialmente, si el programa en cuestión no ha sido probado en Linux o depende de otro software que quizá no se encuentre instalado.

5.1. Actualización de bibliotecas

La mayoría de programas en un sistema Linux se compilan para utilizar bibliotecas compartidas. Estas bibliotecas contienen funciones útiles comunes a muchos programas. En lugar de guardar una copia de estas rutinas en cada programa que las llama, las bibliotecas se encuentran en los archivos del sistema, legibles por todos los programas en el momento de su ejecución. Es decir, cuando se ejecuta un programa, se lee el código desde el propio archivo del programa seguido por cualquier rutina de los archivos de bibliotecas compartidas. Así se ahorra mucho espacio en disco, ya que sólo se guarda en el disco una copia de las rutinas de la biblioteca.

Si tenemos suerte, utilizar el sistema de paquetes significa que las versiones correctas de la biblioteca que necesita cada programa están instaladas junto con dichos programas. El sistema de paquetes se supone que es consciente del cumplimiento de las dependencias de las bibliotecas compartidas. Pero como los distintos programas pueden depender de distintas versiones de bibliotecas o como podemos instalar un programa sin utilizar el sistema de paquetes, ocasionalmente tendremos que conocer los convenios de bibliotecas utilizados en esta sección. Algunas veces, es necesario compilar un programa para que tenga su propia copia de rutinas de bibliotecas, normalmente para la depuración, en lugar de utilizar las rutinas de las bibliotecas compartidas. Estos programas creados de esta manera se dice que están vinculados estáticamente, mientras que los programas creados para utilizar bibliotecas compartidas se dice que están vinculados dinámicamente.

Por consiguiente, los ejecutables vinculados dinámicamente dependen de la presencia de bibliotecas compartidas en el disco duro. Las bibliotecas compartidas se implantan de forma que los programas compilados para utilizarlas, no dependan estrictamente de la versión de las bibliotecas disponibles, lo que significa que podemos actualizar nuestras bibliotecas compartidas y todos los programas creados para utilizar dichas bibliotecas utilizarán automáticamente las nuevas rutinas. Existe una excepción: si se producen cambios en una biblioteca principal, los programas antiguos que no tengan el soporte necesario para utilizar la nueva versión de la biblioteca principal no funcionarán con ella. Sabremos que este es el caso porque el número de versión de la biblioteca principal es diferente. En este caso, tenemos que mantener tanto las antiguas como las nuevas versiones de bibliotecas. Todos los ejecutables antiguos continuarán utilizando las bibliotecas antiguas y cualquier programa nuevo compilado utilizará las nuevas bibliotecas.

Cuando montamos un programa para utilizar bibliotecas compartidas, se añade un fragmento de código al programa que lo obliga a ejecutar *ld.so*, el enlazador dinámico; cuando se inicia el programa. *ld.so* es también el responsable de buscar las bibliotecas compartidas que el programa necesita y de cargar las rutinas en memoria. Los programas vinculados dinámicamente también están vinculados a las rutinas *auxiliares (stub)*, que simplemente toman el lugar de las rutinas de la biblioteca compartida real en el ejecutable. *ld.so* reemplaza la rutina auxiliar con el código de las bibliotecas cuando se ejecuta el programa.

Para listar las bibliotecas compartidas de las que depende un determinado ejecutable podemos utilizar el comando *ldd*:

```
$> ldd /bin/cat
linux-gate.so.1 => (0xffffe000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7e07000)
/lib/ld-linux.so.2 (0xb7f59000)
```

Aquí podemos observar que el programa *cat* depende de tres bibliotecas compartidas incluyendo: *linux-gate*, *libc* y *ld-linux*. Por lo general las bibliotecas que empiezan por *libX*, así

como las que empiezan con *libSM* y *libICE* están todas relacionadas con el sistema *X Window*; *libc* es la biblioteca estándar de C. También podemos ver los números de versión de las bibliotecas para la que se ha compilado el programa, es decir, la versión de las rutinas auxiliares utilizadas. Así como también el nombre del archivo que contiene cada una de las bibliotecas compartidas. Éste es el archivo que buscara *ld.so* cuando se ejecute el programa. Por cierto, el primer archivo de la lista, *linux-gate.so.1*, no es una biblioteca compartida real sino un objeto dinámico compartido proporcionado por el núcleo, un detalle técnico que agiliza las llamadas del sistema en el núcleo y proporciona otros elementos útiles de bajo nivel.

Para utilizar una biblioteca compartida, la versión de las rutinas auxiliares en el ejecutable tiene que ser compatible con la versión de las bibliotecas compartidas. Básicamente, una biblioteca es compatible si la versión del número principal coincide con la versión de las rutinas auxiliares. El número de versión principal es la parte derecha que se encuentra tras *.so*. En este caso, *libc* se utiliza con la versión principal 6. El archivo de biblioteca *libc.so.6*, que normalmente reside en el directorio */lib/tls/i686/cmov/*, puede ser sólo un vínculo simbólico, por ejemplo, a *libc.so.6.2*, lo que significa que la biblioteca tiene el número de versión principal 6 y el número de versión secundaria 2. Las versiones de bibliotecas con el mismo número de versión principal se supone que son intercambiables. Por consiguiente, si un programa se ha compilado con la versión 6.0 de rutinas auxiliares, el ejecutable puede utilizar las versiones 6.1, 6.2, etc. de la biblioteca compartida. Si se lanzase una nueva versión con el número de versión principal 7 y el número de versión secundaria 3 (y, por ende tuviese el nombre de archivo *libc.so.7.3*), lo único que tendríamos que hacer para utilizar esta nueva versión sería cambiar el vínculo simbólico *libc.so.6.3* par que apunte a la nueva versión. El ejecutable *cat* se beneficiaría así automáticamente de cualquier solución de fallos o similar incluido en la nueva versión.

El archivo */etc/ld.so.conf* contiene una lista de directorios en los que busca *ld.so* para encontrar archivos de bibliotecas compartidas. Un ejemplo del contenido de este archivo sería:

```
$> cat /etc/ld.so.conf
/usr/lib
/usr/local/lib
/usr/X11R6/lib
/opt/kde3/lib
```

ld.so también busca en */lib* y */usr/lib*, independientemente del contenido de */etc/ld.so.conf*. Normalmente no existe ninguna razón para modificar este archivo y la variable de entorno *LD_LIBRARY_PATH* puede añadir directorios adicionales a esta ruta de acceso de búsqueda, por ejemplo, si tenemos nuestras propias bibliotecas compartidas privadas que no se deben utilizar a nivel del sistema. Sin embargo, si añadimos entradas al archivo */etc/ld.so.conf* o actualizamos o instalamos bibliotecas adicionales en nuestro sistema, debemos asegurarnos de utilizar el comando *ldconfig*, el cual se encargará de regenerar la caché de la biblioteca compartida en el archivo */etc/ld.so.cache* de la ruta de acceso de búsqueda de *ld.so*. Esta caché la utiliza *ld.so* para encontrar bibliotecas rápidamente en el momento de ejecución sin tener que buscar las direcciones en su ruta de acceso.

Ahora que ya sabemos cómo se utilizan las bibliotecas compartidas, vamos a analizar como actualizarlas. Las dos bibliotecas que se actualizan con más frecuencia son *libc*, la biblioteca estándar de C, y *libm*, la biblioteca de matemáticas. Como designarlas es un tanto especial, vamos a examinar aquí otra biblioteca, *libncurses*, que *emula* un sistema gráfico de ventanas en la consola de texto.

Existen dos archivos independientes para cada una de las bibliotecas compartidas:

- **<biblioteca>.a**: Ésta es la versión estática de la biblioteca. Cuando un programa está vinculado estáticamente, las rutinas se copian directamente desde este archivo en el ejecutable, para que el ejecutable contenga su propia copia de las rutinas de biblioteca. En algunas distribuciones, las versiones estáticas de las bibliotecas se mueven en un paquete independiente y no tiene que estar instalado necesariamente de forma predeterminada.
- **<biblioteca>.so.<versión>**: Ésta es la propia imagen de la biblioteca compartida. Cuando un programa está vinculado dinámicamente, las rutinas auxiliares de este archivo se copian en el ejecutable, permitiendo a *ld.so* localizar la biblioteca compartida en el momento de la ejecución. Cuando el programa se ejecuta, *ld.so* copia las rutinas de la biblioteca compartida en memoria para su uso por el programa. Si un programa está vinculado dinámicamente, no se utiliza el archivo **<biblioteca>.a** para esta biblioteca.

La biblioteca *libncruses* tendrá archivos como: *libncruses.a* y *libncruses.so.5.4*. Los archivos *.a* normalmente se alojan en el directorio */usr/lib*, y los archivos *.so* en */lib*. Cuando compilamos un programa, se utiliza el archivo *.a* o el archivo *.so* para la vinculación y el compilador busca en el directorio */lib* y */usr/lib* (así como en otros lugares) de forma predeterminada. Si tenemos nuestras propias bibliotecas, podemos tener estos archivos en cualquier otra parte y controlar dónde debe buscar el enlazador, con la opción *-L* para el compilador.

Para la mayoría de bibliotecas del sistema, la imagen de la biblioteca compartida, **<biblioteca>.so.<versión>**, se aloja en el directorio */lib*. Las imágenes de bibliotecas compartidas se pueden encontrar en cualquier directorio de búsqueda de *ld.so* en el momento de la ejecución; estos directorios incluyen */lib*, */usr/lib* y los directorios listados en el archivo */etc/ld.so.conf*.

Si examinamos el directorio */lib*, podremos ver una colección de archivos como la siguiente:

```
$> ls -l /lib
lrwxrwxrwx 1 root root 17 Jul 11 06:45 /lib/libncruses.so.5 -> libncruses.so.5.4
-rwxr-xr-x 1 root root 319472 Jul 11 06:45 /lib/libncruses.so.5.4
lrwxrwxrwx 1 root root 13 Jul 11 06:45 libz.so.1 -> libz.so.1.2.2
-rwxr-xr-x 1 root root 62606 Jul 11 06:45 libz.so.1.2.2
...
```

Aquí podemos ver imágenes de biblioteca compartida par dos bibliotecas: *libncruses* y *libz*. Debemos tener en cuenta que cada imagen tiene un vínculo simbólico, denominado **<biblioteca>.so.<major>**, siendo **<major>** el número de la versión principal de la biblioteca. El número secundario se omite ya que *ld.so* busca una biblioteca sólo por su número de versión principal. Cuando *ld.so* ve un programa que ha sido compilado con las rutinas auxiliares para la versión 5.4 de *libncruses*, buscará un archivo denominado *libncruses.so.5* en su ruta de búsqueda. Aquí, */lib/libncruses.so.5* es un vínculo simbólico a */lib/libncruses.so.5.4*, la versión real de la biblioteca que hemos instalado. Al actualizar una biblioteca, tenemos que reemplazar los archivos *.a* y *.so.<versión>* correspondientes a dicha biblioteca. Reemplazar el archivo *.a* es muy fácil: sólo tenemos que copiarlo con las nuevas versiones. Sin embargo, tendremos que tomar algunas precauciones cuando reemplacemos la imagen de la biblioteca compartida, *.so.<versión>*; muchos de los programas basados en texto en el sistema dependen de las imágenes de biblioteca compartida, por lo que no podemos simplemente eliminarlas ni cambiar su nombre. Es decir, el vínculo simbólico **<biblioteca>.so.<major>** siempre tiene que apuntar a una imagen de biblioteca válida. Para ello, debemos copiar primero la nueva imagen a */lib* y

después cambiar el vínculo simbólico para que apunte al nuevo archivo utilizando el comando `ln -sf`. El siguiente ejemplo es una demostración de esta tarea.

Supongamos que estamos realizando una actualización desde la versión 5.4 de la biblioteca *libncruses* a la versión 5.5. Debemos tener los archivos *libncruses.a* y *libncruses.so.5.5*. Lo primero que debemos hacer es copiar el archivo *.a* a la ubicación apropiada, sobrescribiendo la antigua versión:

```
$> sudo cp libncruses.a /usr/lib
```

Ahora debemos copiar el archivo de la nueva imagen en el directorio */lib*; o en el directorio donde se encuentre la imagen de la biblioteca:

```
$> sudo cp libncruses.so.5.5 /lib
```

Ahora, si utilizamos el comando `ls -l /lib`, deberíamos ver una salida parecida a la siguiente:

```
$> ls -l /lib
lrwxrwxrwx 1 root root 17 Jul 11 06:45 /lib/libncruses.so.5 -> libncruses.so.5.4
-rwxr-xr-x 1 root root 319472 Jul 11 06:45 /lib/libncruses.so.5.4
-rwxr-xr-x 1 root root 321042 Jul 11 06:45 /lib/libncruses.so.5.5
...
```

Para actualizar el vínculo simbólico de tal manera que ahora apunte a la nueva biblioteca, debemos utilizar el siguiente comando:

```
$> sudo ln -sf /lib/libncruses.so.5.5 /lib/libncruses.so.5
```

Cuyo resultado será:

```
$> ls -l /lib
lrwxrwxrwx 1 root root 17 Jul 11 06:45 /lib/libncruses.so.5 -> libncruses.so.5.5
-rwxr-xr-x 1 root root 319472 Jul 11 06:45 /lib/libncruses.so.5.4
-rwxr-xr-x 1 root root 321042 Jul 11 06:45 /lib/libncruses.so.5.5
...
```

Ahora podemos eliminar con seguridad el archivo antiguo de imagen, *libncruses.so.5.4*. Es estrictamente necesario utilizar el comando `ln -sf` para reemplazar el vínculo simbólico en un solo paso, especialmente cuando estemos actualizando bibliotecas circulares, como *libc*. Si elimináramos primero el vínculo simbólico y después intentáramos utilizar `ln -s` para añadirlo de nuevo, es muy probable que el comando `ln` no pueda ejecutarse ya que el vínculo simbólico ya no existe; y en lo que respecta a *ld.so*, no podrá encontrar la biblioteca *libc*. Una vez eliminado el vínculo simbólico, casi todos los programas de nuestro sistema no podrán ejecutarse. Debemos tener mucho cuidado cuando actualicemos imágenes de bibliotecas compartidas. Para *libncruses*, las acciones son menos importantes ya que siempre dispondremos de programas de línea de comandos para limpiar cualquier desorden que se haya organizado, pero si estamos acostumbrados a utilizar programas basados en *ncruses*, como *Midnight Commander*, podría ser un inconveniente.

Siempre que actualicemos o agreguemos una biblioteca al sistema, es recomendable ejecutar *ldconfig* para regenerar la caché de la biblioteca utilizada por *ld.so*. En algunos casos nos puede ocurrir que *ld.so* no nos reconozca una nueva biblioteca hasta que no ejecutemos *ldconfig*.

Ahora sólo nos queda una pregunta: ¿dónde podemos obtener las versiones nuevas de las bibliotecas? Podemos descargar diversas versiones de bibliotecas básicas del sistema desde el directorio <ftp://ftp.gnu.org/pub/gnu/glibc>. Este directorio contiene versiones de fuentes y bibliotecas relacionadas. Otras bibliotecas se mantienen y archivan de forma independiente. De todas formas, todas las bibliotecas instaladas incluirán los archivos de la versión *.so*, y, posiblemente, los archivos *.a*, así como un conjunto de archivos *include* para su uso con el compilador.

TEMA 4: INICIO Y CIERRE DEL SISTEMA

Objetivos

- Diferenciar los métodos más comunes para arrancar un sistema Linux.
- Estudiar las características funcionales y algunas opciones de configuración del gestor de arranque *GRUB*.
- Comprender gran parte de los mecanismos que subyacen en el proceso de inicio de un sistema Linux.
- Conocer algunos archivos y directorios en donde se encuentran los comandos de inicio del sistema.
- Comprender la necesidad de llevar a cabo un cierre seguro dentro de un sistema Linux.

Contenido

1. Inicio del sistema
 - 1.1. Disquete de arranque
 - 1.2. El gestor de arranque *GRUB*
 - 1.2.1. El archivo */etc/grub.conf*
 - 1.2.2. Especificar las opciones del arranque
 - 1.2.3. Eliminar *GRUB*
2. Inicio e inicialización del sistema
 - 2.1. Mensajes de inicio de núcleo
 - 2.2. Archivos *init*, *inittab* y *rc*
 - 2.3. Archivos *rc*
3. Modo de un solo usuario
4. Cierre del sistema

Bibliografía

Básica

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.

Complementaria

- Roger Baig Viñas, Francesc Aulí Llinàs, “Sistema operativo GNU/Linux básico Primera Edición”, UOC Formación de Posgrado, Software libre, noviembre 2003.
http://www.nodo50.org/cursos/manual_gnu_linux.pdf
- Josep Jorba Esteve, Remo Suppi Boldrito, “Administración avanzada de GNU/Linux Primera Edición”, UOC Formación de Posgrado, Software libre, 2004.
<http://www.uoc.edu/masters/esp/img/871.pdf>

1. Inicio del sistema

Existen diversos métodos para iniciar Linux en nuestro sistema. Los métodos más comunes implican arrancar desde el disco duro o desde un disquete de arranque. En muchas ocasiones, el procedimiento de instalación habrá configurado uno de estos métodos. En cualquier caso siempre es importante saber cómo configurar el arranque de nuestro sistema.

1.1. Disquete de arranque

Tradicionalmente, un disquete de arranque Linux simplemente contiene una imagen del núcleo que se carga en memoria al insertar el disquete e iniciar el sistema. Un disquete de arranque Linux puede contener en su lugar un registro de arranque *GRUB*, que hace que el sistema se inicie a un núcleo desde el disco duro. Muchas distribuciones de Linux crean un disquete de arranque automáticamente al instalar el sistema. El uso de un disquete de arranque es un método muy fácil para iniciar Linux, si no deseamos iniciarlo desde el disco duro. Tras iniciar el núcleo desde el disquete de arranque, podremos utilizar la disquetera para otros propósitos.

En esta sección vamos a incluir alguna información técnica para explicar el proceso de arranque de nuestro sistema Linux, todo esto para conseguir un mejor entendimiento y comprensión de todos los factores que se ven involucrados en el arranque del sistema.

Normalmente, la imagen del núcleo está comprimida, utilizando el mismo algoritmo que utilizan los programas de compresión como *gzip* o *bzip2*. La compresión permite que el núcleo, que puede tener varios megabytes o más de tamaño, requiera sólo unos cientos de kilobytes de espacio en disco. Parte del código del núcleo no está comprimido: esta parte es la que contiene las rutinas necesarias para descomprimir el núcleo desde la imagen del disco y cargarlo en memoria. Por consiguiente, el núcleo realmente se inicia asimismo en el momento del arranque mediante su descompresión en memoria.

En la imagen del núcleo se guardan diversos parámetros. Entre estos parámetros se encuentra el nombre del dispositivo a utilizar como sistema de archivos *raíz* o principal cuando se inicia el núcleo. Otro parámetro es el modo de texto a utilizar para la consola del sistema. Todos estos parámetros se pueden modificar utilizando el comando *rdev*.

Una vez iniciado el núcleo, éste intenta montar un sistema de archivos en el dispositivo principal incrustado en la propia imagen del núcleo, que servirá como sistema de archivos principal, es decir, el sistema de archivos en */*. Aquí lo principal es que la imagen del núcleo tiene que contener el nombre correcto del dispositivo del sistema de archivos principal. Si el núcleo no puede montar un sistema de archivos en este dispositivo, desiste de ejecutar la acción y emite un mensaje de *kernel panic*. Básicamente un mensaje de *kernel panic* es un error fatal señalado por el propio núcleo. Un *kernel panic* se producirá siempre que el núcleo este confundido y no pueda continuar con la ejecución. Por ejemplo, si existe un fallo en el propio núcleo, se podría producir un *kernel panic* cuando el núcleo intente acceder a una memoria que no existe.

El dispositivo principal guardado en la imagen del núcleo es de nuestro sistema de archivos principal en el disco duro, lo que significa que cuando se inicie el núcleo, montará una partición de disco duro como sistema de archivos principal y todo el control se transferirá al disco duro. Una vez cargado el núcleo en memoria, permanecerá ahí; ya no será necesario acceder nuevamente al disquete de arranque, a no ser que reiniciemos el sistema, por supuesto.

Dada una imagen de núcleo razonablemente pequeña, podemos crear nuestro propio disquete de arranque. En muchos sistemas de Linux, el propio núcleo se guarda en el archivo */boot/vmlinuz*. ¿Por qué este nombre de archivo tan extraño? En muchos sistemas Unix, el

núcleo se guardaba en un archivo denominado `/vmunix`, siendo *vm* un acrónimo de las palabras inglesas utilizadas para referirse a la memoria virtual (*virtual memory*). Naturalmente, Linux tiene que ser diferente y denomina a sus imágenes de núcleo *vmlinux*, colocándolas en el directorio `/boot` para extraerlas del directorio *raíz*. El nombre *vmlinuz* se adoptó para diferenciar las imágenes del núcleo comprimidas de las imágenes del núcleo sin comprimir. La denominación *vmlinuz* no es ningún convenio universal; sin embargo, otros sistemas de Linux guardan el núcleo en `/vmlinuz` o en `/vmlinux`, e incluso otros en un archivo como `/Image`. Debemos tener en cuenta que los sistemas Linux instalados recientemente pueden no tener una imagen del núcleo en el disco duro, si se ha creado automáticamente un disquete de arranque. En cualquier caso, podemos montar nuestro propio núcleo. Normalmente es recomendable hacerlo en cualquier caso: podemos personalizar el núcleo para que incluya sólo los controladores para el hardware de nuestra máquina en concreto.

Vamos a suponer que tenemos una imagen del núcleo en el archivo `/boot/vmlinuz`. Para crear un disquete de arranque, el primer paso es utilizar el comando `rdev` para establecer el dispositivo principal en el sistema de archivos principal de Linux. Si montamos en núcleo manualmente, esta opción debería estar establecida en el valor correcto.

Como usuario `root`, debemos utilizar `rdev -h` para imprimir un mensaje sobre la utilización del comando. Hay muchas opciones que nos permiten especificar el dispositivo *root*, el dispositivo de intercambio, el disco *RAM*, etc.

Si utilizamos el comando `rdev /boot/vmlinuz`, se imprimirá del dispositivo principal codificado en el núcleo que se encuentra en `/boot/vmlinuz`:

```
$> sudo rdev /boot/vmlinuz
Root device /dev/hda1
```

Si la información obtenida con el comando anterior no es correcta y el sistema de archivos principal se encuentra en, por ejemplo en `/dev/hda3`, debemos utilizar el siguiente comando:

```
$> sudo rdev /boot/vmlinuz /dev/hda3
```

`rdev` es un comando silencioso; no se imprime nada cuando establecemos el dispositivo principal por lo que debemos ejecutar de nuevo `sudo rdev /boot/vmlinuz` para comprobar que los cambios han sido establecidos correctamente.

Ahora estamos preparados para crear el disquete de arranque. Para obtener los mejores resultados es recomendable utilizar un disquete nuevo y recién formateado. Podemos formatear el disquete en Linux mediante el uso del comando `sudo fdformat /dev/fdX` (donde la *X* indica el número de nuestra unidad de disquete asignada por nuestro Linux); así podremos fijar la información del sector y de la pista para que el sistema pueda detectar automáticamente el tamaño del disquete.

Para crear el disquete de arranque, debemos utilizar el comando `dd` para copiar en él la imagen del núcleo, como en el siguiente ejemplo:

```
$> sudo dd if=/boot/vmlinuz of=/dev/fd0 bs=8192
```

Este comando copia el archivo de entrada (*if*, *input file*) denominado `/boot/vmlinuz` en el archivo de salida (*of*, *output file*) denominado `/dev/fd0` (el primer dispositivo de disquetes en nuestro sistema), utilizando un tamaño de bloque (*bs*, *block size*) de 8192 bytes. Evidentemente también podríamos haber utilizado el comando `cp` para realizar esta tarea.

Ahora nuestro disquete de arranque está preparado para su uso. Podemos cerrar nuestro sistema y arrancar con el disquete y, si todo va bien, nuestro sistema Linux debería arrancar como lo hace normalmente. Es recomendable crear un disquete de arranque de repuesto por si alguna vez las cosas nos llegan a fallar.

Cabe destacar que hoy en día la utilidad de los disquetes se ha visto reemplazada por los dispositivos de almacenamiento ópticos como los CD o los DVD, y es por eso que la gran mayoría de distribuciones nos permiten utilizar el mecanismo de los *live-CD*, a parte de utilizarlos para instalar el sistema, para hacer uso de ellos como CD o DVD de arranque, sólo nos bastará con introducir el CD o DVD y arrancar desde él, para luego montar nuestro sistema de archivos ubicado en el disco duro y llevar a cabo las tareas de mantenimiento o reparación sobre él. Hoy en día también se están utilizando mucho los dispositivos USB de almacenamiento (pendrives, memory sticks, compact flash, etc.) y son tan versátiles que también los podemos utilizar para crear una llave USB, como se les suele decir, para conseguir arrancar un sistema Linux desde ellos, para llevar a cabo con ellos las mismas tareas de mantenimiento y reparación que con los *live-CD*.

1.2. El gestor de arranque GRUB

GRUB (GRand Unified Bootloader) es un gestor de arranque de propósito general que puede arrancar cualquier sistema operativo instalado en una máquina, incluyendo Linux. Existen docenas de métodos para configurar *GRUB*. Aquí vamos a analizar los dos métodos más comunes: instalar *GRUB* en el registro de arranque maestro de nuestro disco duro e instalar *GRUB* como gestor de arranque secundario sólo para Linux.

Es importante hacer énfasis en que *GRUB* no es el único gestor de arranque disponible para iniciar Linux. Existen otras alternativas, como LILO (*Linux LOader*) que funcionan igual de bien.

GRUB es el método más común para iniciar Linux desde el disco duro. Cuando nos referimos a *arrancar desde el disco duro* queremos decir que el propio núcleo se guarda en el disco duro y que no necesitamos de ningún dispositivo (disquete, CD, DVD, etc.) de arranque para cargarlo, pero debemos recordar que aunque utilicemos un disquete de arranque, el control se transfiere al disco duro cuando el núcleo se carga en memoria. Si *GRUB* está instalado en el registro de arranque maestro (*MBR, Master Boot Record*), es el primer código en ejecutarse cuando se arranca el disco duro. *GRUB* puede arrancar posteriormente cualquier sistema operativo, como Linux o Windows, y permitirnos seleccionar entre ellos en el momento del arranque.

Windows NT y las versiones posteriores de Windows (2000/XP/Vista) tienen gestores de arranque de su propiedad que se apoderan del *MBR*. Si estamos utilizando uno de estos programas, para poder arrancar Linux desde el disco duro, puede que tengamos que instalar *GRUB* como gestor de arranque *secundario* sólo para Linux. En este caso, *GRUB* se instala en el registro de arranque sólo para nuestra partición principal de Linux y el software del gestor de arranque, para Windows NT/2000/XP/Vista, se hace cargo de ejecutar *GRUB* desde ahí cuando deseemos iniciar Linux.

Sin embargo, como podremos comprobar, los gestores de arranque de Windows NT/2000/XP/Vista son, de algún modo, poco cooperativos cuando se trata de arrancar *GRUB*. Se trata de una mala decisión de diseño; y si tenemos que utilizar uno de estos gestores de arranque inevitablemente, quizá sea más fácil iniciar Linux desde el disquete de arranque en su lugar. O, si realmente deseamos seguir con Linux, podemos utilizar *GRUB* para arrancar Windows NT/2000/XP/Vista y descargar totalmente los gestores de arranque de Windows. Este método normalmente es muy fácil y es el más recomendado. También es el que instalan

automáticamente la mayoría de distribuciones si intentamos instalar Linux en un equipo que tiene una instalación de Windows existente.

Utilizar *GRUB* con Windows NT/2000/XP/Vista es muy fácil. Sólo tenemos que configurar *GRUB* para que inicie Windows NT/2000/XP/Vista. Sin embargo, si instalamos Windows NT/2000/XP/Vista tras la instalación de *GRUB* o en si de Linux, tendremos que volver a instalar o intentar recuperar *GRUB*; ya que el procedimiento de instalación de Windows NT/2000/XP/Vista sobrescribe el *MBR* del disco duro principal. Sólo tenemos que asegurarnos de que tenemos a mano un disquete de arranque de Linux para poder iniciar Linux y volver a ejecutar *GRUB*.

Debido a que la gran mayoría de sistemas modernos carecen de unidades de disquetes, se ha desarrollado un sistema de recuperación de *GRUB* basada en CD, esta recibe el nombre de *SuperGRUB* (<http://supergrub.forjamari.linux.org/>) y nos permite iniciar con un CD de arranque el cual consigue recuperar el *GRUB* de nuestro Linux tras una serie sencilla de pasos que le hemos de indicar.

Debemos tener en cuenta que la gran mayoría de distribuciones modernas de Linux configuran e instalan *GRUB* cuando instalamos por primera vez todo el sistema Linux. Es una buena idea dejar simplemente que el programa de instalación de nuestra distribución instale automáticamente *GRUB* y comprobar posteriormente por nuestra cuenta lo que ha hecho; así tendremos un punto de inicio para poder ajustar posteriormente *GRUB* a nuestras necesidades.

1.2.1. El archivo */etc/grub.conf*

El primer paso en la configuración de *GRUB* es establecer nuestro archivo de configuración, que normalmente se guarda en */etc/grub.conf*. El archivo */etc/grub.conf* hace referencia a otros archivos de configuración que examinaremos más adelante. Normalmente, el archivo */etc/grub.conf* suele ser muy corto.

Debemos señalar que *GRUB* es muy flexible y nos permite introducir comandos *GRUB* de forma interactiva durante el proceso de arranque. Sin embargo, la gran mayoría considera esto como algo tedioso y propenso al error, razón por la que aquí describiremos otro uso, que nos proporcionará un menú conveniente desde donde podemos elegir, por ejemplo, iniciar dos núcleos diferentes o incluso sistemas operativos diferentes. Éste es un ejemplo de un archivo */etc/grub.conf* que posee una configuración bastante concisa:

```
root (hd0, 0)
install --stage2=/boot/grub/stage2 /grub/stage1 (hd0) /gurb/stage2 0x8000
(hd0, 0)/grub/menu.lst
quit
```

La primera línea especifica la unidad desde donde se efectúa el arranque. En este caso, es la primera partición en el primer disco duro. *hd* es *hard drive*, el primer cero es el primer disco duro en el sistema y el segundo cero es la primera partición en ese determinado disco duro; ¡*GRUB* siempre empieza a contar desde cero!

Éstos son algunos ejemplos más: (*fd0*) significaría arrancar desde el primer disquete de arranque en el sistema y (*hd3, 4*) significaría la quinta partición en el cuarto disco duro. También hay un nombre especial que es, (*nd*), que se utiliza para iniciar la imagen del núcleo desde la red.

La segunda línea es bastante compleja. Tardaríamos mucho en detallar el proceso de arranque completo, pero, al menos, podemos decir que *GRUB* utiliza un proceso de arranque de

dos fases y este comando especifica dónde se obtienen las instrucciones para las dos fases, en qué dirección se deben cargar en la memoria del equipo y qué es lo que hay que hacer a continuación. La parte relacionada a: que es lo que hay que hacer a continuación es la más interesante para nosotros; en el archivo de configuración de ejemplo, es la parte (*hd0, 0*)/*grub/menu.lst* del final.

Esta línea hace referencia a un archivo que utiliza *GRUB* que se encuentra en la carpeta de arranque (*/boot*) de un dispositivo pues el uso de los paréntesis lo indica así, en el caso del ejemplo una partición de disco duro, la primera partición en el primer disco duro.

```
$> ls /boot/grub/menu.lst
/boot.grub/menu.lst
```

El siguiente es un ejemplo del contenido del archivo */boot/grub/menu.lst* que carga dos configuraciones de Linux y MS-Windows diferentes, presentado en un menú conveniente:

```
default 0
timeout 10
```

```
title Linux
kernel (hd0, 5)/boot/vmlinuz root=/dev/hda6 vga=0x314
```

```
title Linux Failsafe
kernel (hd0, 5)/boot/vmlinuz root=/dev/hd6 ide=nodma apm=off acpi=off
vga=normal
noresume barrier=off nosmp noapic maxcpus=0 3
```

```
title Windows
root(hd0, 0)
chainloader +1
```

Las dos primeras líneas juntas indican que cuando se presente el menú de inicio a través de *GRUB*, el usuario tendrá diez segundos (*timeout*) para realizar una elección; en caso contrario, se cargará la primera entrada de la siguiente lista.

Tras estas dos líneas iniciales, hay tres secciones que empiezan cada una de ellas por la cláusula *title*. Tras *title*, se especifica una cadena que se va a mostrar en el menú de inicio en el momento del arranque. Para las dos configuraciones de Linux, existe a continuación una línea de núcleo que muestra desde dónde se ha cargado la imagen del núcleo. El resto se pasa directamente al núcleo como parámetros de arranque, incluyendo cuál será el dispositivo principal y el modo de la terminal (*vga=0x314*). En la denominada configuración de protección contra fallos (*failsafe*), especificamos muchos parámetros del núcleo que se activan igual que cualquier otra actividad del núcleo que tenga una pequeña oportunidad de ir mal. Un sistema como éste será lento y no tendrá una funcionalidad completa, pero si hemos configurado mal algo, el núcleo de protección contra fallos seguirá adelante y nos permitirá arrancar el sistema, al menos, para poder reparar los errores. El comando *kernel* carga el núcleo en memoria, pero no lo arranca realmente; el comando *boot* ejecuta el proceso real de arranque. Sin embargo, en el sistema de menú *GRUB* de nuestro ejemplo, el comando *boot* está implícito, pero perfectamente puede ser escrito al final de cada sección.

La carga en Windows funciona de forma diferente. *GRUB* no puede cargar directamente otros sistemas operativos que no sean Linux, la familia BSD y algunos otros. Para dichos sistemas, como Windows, *GRUB* llama en su lugar al gestor de arranque incluido en cada sistema, acción que es denominada carga en cadena, y el comando *GRUB* que realiza esta

acción es el comando *chainloader*. La opción *+l* significa que *GRUB* puede encontrar el gestor de arranque en la partición especificada con el comando *root* anterior, un sector en la partición.

Cuando estemos satisfechos con nuestra configuración de *GRUB*, tendremos que instalarlo. Esta tarea se lleva a cabo con el comando *grub-install*, que espera a que se le indique el directorio en el que se encuentran los archivos de fase y la imagen del núcleo; y en qué dispositivo tiene que instalar el gestor de arranque. Para ello, podemos utilizar el siguiente comando:

```
$> sudo grub-install --root-directory=/boot /dev/hda
```

Este código instala el gestor de arranque en el primer disco duro *IDE* (*/dev/hda*), que normalmente es el que busca la *BIOS* del equipo para encontrar la información de arranque inicial.

1.2.2. Especificar las opciones del arranque

Cuando instalamos por primera vez Linux, es muy probable que lo arranquemos desde un disquete o CD-ROM que nos proporcione un menú de arranque de *GRUB*, u otro gestor de arranque, si seleccionamos una entrada y escribimos *e*, obtendremos un indicador de comandos de inicio. En este indicador podemos introducir diversas opciones para el arranque, como:

```
hd=cylinders,heads,sectors
```

Esto nos permite especificar la geometría de nuestro disco duro. Cada vez que inicie Linux, puede que tengamos que especificar estos parámetros para que nuestro hardware se detecte correctamente. Si estamos utilizando *GRUB* para iniciar Linux desde el disco duro, podemos especificar dichos parámetros en la línea *kernel* en el archivo de configuración de *GRUB* en lugar de introducirla en el indicador de comandos del inicio cada vez. Para la entrada de Linux, sólo tenemos que añadir una línea como la siguiente:

```
append = "hd=683,16,38"
```

Así, el sistema se comportara como si *hd=683,16,38* se hubiese introducido en la línea de comandos de inicio de *GRUB*. Si deseamos especificar múltiples opciones de arranque, podemos hacerlo con una sola línea *append* como:

```
append = "hd=683,16,38 hd=64,32,202"
```

Con este ejemplo, especificamos la geometría para el primer y segundo disco duro respectivamente. Cuando hayamos realizado los cambios en el indicador de comandos de inicio, debemos presionar la tecla *Esc* para volver atrás al menú de inicio y arrancar el sistema desde ahí.

Debemos tener en cuenta que tenemos que utilizar estas opciones de arranque sólo si el núcleo no detecta nuestro hardware en el momento del arranque, algo poco probable a no ser que tengamos un hardware muy antiguo o poco común.

Existen otras opciones para el momento del arranque. La mayoría de ellas tratan la detección del hardware. Sin embargo, las siguientes opciones adicionales también pueden resultarnos útiles:

- **single:** Arranque el sistema en modo de un solo usuario; omite toda la configuración del sistema e inicia una *shell* de *root* en la consola.

- **root=partición:** Monta la partición denominada como sistema de archivos de *root* de Linux.
- **ro:** Monta el sistema de archivos como de sólo lectura. Normalmente se utiliza para ejecutar *fsck*.
- **ramdisk=size:** Especifica un tamaño, en bytes, para el dispositivo de disco de memoria RAM; principalmente es útil para la instalación.
- **vga=mode:** Establece el modo de pantalla VGA. Los modos válidos son: *normal*, *extended*, *ask* o un entero.
- **mem=size:** Le indica al núcleo la cantidad de RAM que tenemos. Si disponemos de 64MB o menos, el núcleo puede obtener esta información del *BIOS*, pero si utilizamos un núcleo más antiguo y disponemos de más memoria, tendremos que indicarle al núcleo la cantidad exacta de o sólo utilizará los primeros 64MB. Por ejemplo, si tenemos 128MB, debemos especificar *mem=128m*. Afortunadamente, ya no hay que establecer esta opción en los núcleos más modernos.

Cualquiera de estas opciones se pueden introducir manualmente en el indicador de comandos de inicio de *GRUB* o especificarlas en la línea *kernel* en el archivo de configuración de *GRUB*.

1.2.3. Eliminar *GRUB*

Si por alguna razón extraña hemos decidido dejar de usar nuestro Linux y por ende hemos decidido desinstalarlo totalmente de nuestro ordenador. Podemos dejar de querer utilizar el gestor de arranque *GRUB*. La forma más fácil de eliminarlo es desde Windows utilizando el comando *fdisk* en el símbolo del sistema.

Por ejemplo:

```
C:\> fdisk /MBR
```

Así conseguiremos ejecutar *fdisk* y sobrescribir el *MBR* con un registro de inicio de Windows válido.

2. Inicio e inicialización del sistema

En esta sección vamos a analizar lo que ocurre exactamente cuando se inicia el sistema. Conocer los procesos y los archivos implicados es importante para ejecutar diversos tipos de configuración del sistema.

2.1. Mensajes de inicio de núcleo

El primer paso es iniciar el núcleo. Tal y como lo describimos anteriormente, podemos arrancar el núcleo desde un disquete o desde el propio disco duro. A medida que el núcleo se cargue en la memoria, enviará mensajes a la consola del sistema, pero normalmente también guarda dichos mensajes en archivos de registro del sistema (*logs*). Como *root*, siempre podemos visualizar el contenido del archivo */var/log/messages*, que también contiene mensajes del núcleo emitidos durante el tiempo de ejecución. El comando *dmesg* imprime las últimas líneas del buffer de anillos de mensajes de núcleo; naturalmente, justo tras el arranque, obtendremos los mensajes de inicio mediante dicho comando.

Los siguientes son ejemplos de algunos párrafos sobre un par de mensajes interesantes y lo que significan. Estos mensajes son impresos por el propio núcleo a medida que se inicializa cada controlador de dispositivo. Los mensajes exactos impresos dependerán de los controladores que estén compilados en nuestro núcleo y del hardware que tenga nuestro sistema.

```
Linux versión 2.6.11.4-21.7-default (geeko@builddhost) (gcc version 3.3.5
2005011
7 (prerelease) (SUSE Linux)) #1 Mar Jun 2 14:23:14 UTC 2005
```

Este ejemplo de mensaje nos indica el número de versión del núcleo, la máquina en la que se encuentra y con qué compilador se ha montado.

A continuación, el núcleo informa sobre elementos como el *BIOS*, la cantidad de memoria presente, la configuración de administración de potencia, etc. Éstas son algunas de las líneas más interesantes.

```
...
127MB HIGHMEM available.
896MB LOWMEM available.
...
Kernel command line: root=/dev/hda6 vga=0x314 selinux=0 splash=silent resume=/dev/hda5
...
Detected 599.481 MHz processor.
...
```

A continuación, el núcleo nos indica las configuraciones de consola que él ha escogido y qué tipo de consola ha detectado:

```
Console: color dummy device 80x25
```

Cabe destacar que esta línea implica sólo al modo de texto que está utilizando el núcleo, no a las opciones de tarjeta de vídeo, Tampoco tiene nada que ver con el sistema *X Windows*; el núcleo no está implicado para nada.

Posteriormente podremos observar el cálculo *BogoMIPS* para nuestro procesador:

```
Calibrating delay loop... 1187.84 BogoMIPS (lpj=593920)
```

Ésta es una entrada de medida de la velocidad del procesador totalmente falsa que se utiliza para obtener el rendimiento óptimo en los ciclos de retardo para diversos controladores de dispositivo.

El núcleo recopila información sobre el bus *PCI* y comprueba si hay alguna tarjeta *PCI* presente en el sistema:

```
PCI: PCI BIOS revision 2.10 entry at 0xfd8d6, last bus=8
PCI: Using configuration type 1
...
PCI: Probing PCI hardware (bus00)
PCI: Ignoring BAR0-3 of IDE controller 000:00:1f.1
PCI: Transparent bridge - 0000:00:1e.0
...
```

A continuación, Linux establece el sistema de red, el puerto del ratón y el controlador serie:

```
ttyS00 at 0x03f8 (irq = 4) is a NS16550A
```

Lo anterior significa que se ha detectado el primer dispositivo (*/dev/ttyS00*, o COM1) en la dirección 0x03f8, IRQ 4, utilizando funciones 16550A UART. A continuación se incluyen más detecciones de hardware, como el reloj en tiempo real y la disquetera:

```
Real Time Clock Driver v1.12
```

```
...
```

```
Floppy drive(s): fd0 is 1.44M
```

```
FDC 0 is National Semiconductor PC87306
```

```
loop: loaded (max 8 devices)
```

```
...
```

La línea:

```
Adding 1029632k swap on /dev/hda5. Priority:42 extents:1
```

Indica la cantidad de espacio de intercambio que ha sido encontrada por el núcleo. Entre otras tareas posteriores ejecutadas durante un arranque típico se encuentran la búsqueda y configuración de un puerto paralelo (lp1), la detección y configuración de la tarjeta de red y, por último, la configuración del subsistema USB.

2.2. Archivos *init*, *inittab* y *rc*

Una vez inicializados los controladores de dispositivo, el núcleo ejecuta el programa *init*, que se encuentra en */etc*, */bin* o */sbin*. *init* es un programa de propósito general que produce nuevos procesos y reinicia determinados programas al salir de ellos. Por ejemplo, cada consola tiene un proceso *getty* ejecutándose en ella, iniciada por *init*. Tras el inicio de sesión, el proceso *getty* se reemplaza con otro. Tras salir de la sesión, *init* inicia un nuevo proceso *getty*, permitiéndonos volver a iniciar la sesión de nuevo.

init también es el responsable de ejecutar diversos programas y secuencias de comandos cuando se inicia el sistema. Todo lo que hace *init* se controla a través del archivo */etc/inittab*, pero, para poder entender este archivo, primero tenemos que entender el concepto de nivel de ejecución.

En lo que respecta a *init*, un nivel de ejecución es un número o una letra que especifica un estado actual del sistema. Por ejemplo, cuando un nivel de ejecución del sistema se cambia a tres, se ejecutarán todas las entradas en */etc/inittab* que contengan un tres en la columna que especifica los niveles de ejecución. Los niveles de ejecución son una forma de agrupar entradas en */etc/inittab*. Por ejemplo, puede que deseemos indicar que el nivel de ejecución uno ejecute sólo las secuencias de comandos de configuración mínima, el nivel de ejecución dos todo lo que ejecuta el nivel de ejecución uno más la configuración de red, el nivel de ejecución tres todo lo que ejecutan los niveles uno y dos más un acceso de conexión por marcación telefónica y así sucesivamente. Actualmente, las distribuciones Red Hat y SuSE se han configurado para que el nivel de ejecución cinco inicie automáticamente la interfaz gráfica del sistema *X Windows*. Debian hace lo mismo en los niveles de ejecución dos al cinco.

En general, no tendremos que preocuparnos por los niveles de ejecución. Cuando el sistema se inicia, introduce el nivel de ejecución predeterminado, establecido en */etc/inittab*. En la mayoría de sistemas, este valor predeterminado es tres o cinco. Tras analizar el arranque normal, en la sección anterior, vamos a analizar como introducir otro nivel de ejecución que en ocasiones vamos a necesitar utilizar: el nivel de ejecución uno o modo de un solo usuario.

Vamos a examinar un archivo `/etc/inittab` de ejemplo:

```
# Establecer el nivel de ejecución en tres
id:3:initdefault:

# Primera secuencia de comandos a ejecutar
si::bootwait:/etc/init.d/boot

# Ejecutar /etc/init.d/rc con un nivel de ejecución como argumento
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# Ejecutar al presionar Ctrl+Alt+Supr
ca::ctrlaltdel:/sbin/shutdown -t3 -rf now

# Iniciar agetty para la consolas virtuales desde la uno a la seis
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

Los campos se separan con dos puntos. El último campo es el más reconocible: es el comando que ejecuta `init` para esta entrada. El primer campo es un identificador arbitrario (no importa lo que sea siempre que sea único en el archivo) y el segundo campo indica qué niveles de ejecución hacen que se llame al comando. El tercer campo le indica a `init` cómo tiene que controlar esta entrada; por ejemplo, si tenemos que ejecutar el comando dado una vez o ejecutarlo siempre que aparezca.

El contenido exacto de `/etc/inittab` depende de nuestro sistema y de la distribución de Linux que tengamos instalada.

En nuestro archivo de ejemplo, primero podemos ver que el nivel de ejecución predeterminado está establecido en tres. El campo `action` para esta entrada es `initdefault`, que hace que el nivel de ejecución dado se establezca como predeterminado. Normalmente éste es el nivel de ejecución utilizado cuando se inicia el sistema. Podemos sobrescribir el valor predeterminado con cualquier otro nivel que deseemos ejecutando `init` manualmente y pasándole el nivel de ejecución deseado como un argumento. Por ejemplo, el siguiente comando cierra todos los servicios que pertenecen al nivel de ejecución actual, pero no el nivel de ejecución cinco.

```
$> sudo init 5
```

La siguiente entrada le indica a `init` que ejecute la secuencia de comandos `/etc/init.d/boot` cuando se inicie el sistema. El campo `action` es `[sysinit]`, que especifica que esta entrada debe ejecutarse cuando se inicie por primera vez `init` en el arranque del sistema. El archivo `/etc/init.d/boot` es simplemente una secuencia de comandos de `shell` que contiene órdenes para

controlar la inicialización básica del sistema; por ejemplo, se habilita el espacio de intercambio, se revisan y se montan los archivos del sistema y se sincroniza el reloj con el reloj *CMOS*.

En la siguiente sección vamos a comprobar cómo ejecuta el sistema el archivo */etc/init.d/rc* cuando introducimos cualquiera de los niveles de ejecución hasta el seis, con el nivel de ejecución apropiado como argumento. *rc* es una secuencia de comandos de inicio genérica que ejecuta otras secuencias de comandos apropiadas para dicho nivel de ejecución. Aquí el campo *action* es *wait*, que le indica a *init* que ejecute el comando dado y espere a que termine su ejecución antes de hacer algo más.

2.3. Archivos *rc*

Linux guarda los comandos de inicio en archivos con *rc* en el nombre, utilizando un antiguo convenio de Unix. Los comandos hacen todo lo necesario para que un sistema funcione completamente, como iniciar los servicios o los demonios correspondientes. Gracias a estos comandos, el sistema aparece preparado con opciones de inicio de sesión, correo, un servidor web o cualquier elemento instalado y que se le haya indicado que se ejecute.

En esta sección describiremos la estructura de los archivos *rc* para conocer donde empieza todo y para que podamos iniciar o detener los servicios manualmente por si no hacen lo que nosotros queremos que hagan.

Las secuencias de comandos para cada nivel de ejecución se guardan en el directorio */etc/rcN.d*, siendo *N* el nivel de ejecución que se va a iniciar. Por tanto, para el nivel de ejecución tres, se utilizarán las secuencias de comandos en */etc/rc3.d*.

Si examinamos alguno de estos directorios; podremos observar diversos nombres de archivos como: *Snnxxxx* o *Knnxxxx*, siendo *nn* un número comprendido entre 00 y 99 y *xxxx* el nombre de un servicio del sistema. Las secuencias de comandos cuyos nombres empiezan con *K* se ejecutan primero a través de */etc/init.d/rc* para cerrar cualquier servicio existente y posteriormente se ejecutan las secuencias de comandos cuyos nombres empiezan con *S* para iniciar nuevos servicios.

Los números *nn* en los nombres se utilizan para forzar una ordenación en las secuencias de comandos cuando se ejecutan: las secuencias de comandos con los números más bajos se ejecutan antes que las secuencias que tienen números más altos. El nombre *xxxx* se utiliza simplemente para ayudarnos a identificar qué servicio del sistema se corresponde con la secuencia de comandos. El convenio de denominación puede parecer extraño, pero facilita la adición o eliminación de secuencias de comandos en estos directorios y la ejecución automática en el momento apropiado a través de */etc/init.d/rc*. Para personalizar secuencias de comandos de inicio, es muy cómodo utilizar un editor gráfico de niveles de ejecución, como *KSysV* en entornos de escritorios de *KDE* o *BUM* en entornos de escritorios de *GNOME*. Algunas distribuciones incluyen por defecto un editor gráfico de niveles de ejecución como parte de sus herramientas de administración.

Por ejemplo, la secuencia de comandos que inicializa el sistema de red se puede denominar *S10network*, mientras que la secuencia de comandos que detiene el demonio de inicio de sesión del sistema se puede denominar *K70login*. Si se colocan dichos archivos en los directorios */etc/rcN.d* apropiados, */etc/init.d/rc* los ejecutará en orden numérico, durante el inicio del sistema o en el momento del cierre. Si el nivel de ejecución predeterminado de nuestro sistema es tres, debemos examinar los archivos contenidos en el directorio */etc/rc3.d* para comprobar qué secuencias de comandos se ejecutan cuando se inicia nuestro sistema normalmente.

Como los mismos servicios se inician o se detienen en niveles de ejecución diferentes, las distintas distribuciones suelen utilizar vínculos simbólicos en lugar de repetir la misma

secuencia de comandos en múltiples sitios. Por tanto, cada archivo *S* o *K* es un vínculo simbólico que apunta a un directorio central que almacena las secuencias de comandos de inicio o de cierre para todos los servicios, normalmente este directorio que almacena las secuencias de comandos es el directorio */etc/init.d*. Éste directorio contiene una secuencia de comandos denominada *skeleton* que nos permite adaptar los demonios escritos por nosotros mismos para que estos sean iniciados o detenidos.

Es útil conocer la ubicación de una secuencia de comandos de inicio o de cierre en caso de que no deseemos reiniciar completamente o introducir un nivel de ejecución diferente, pero tendremos que iniciar o detener un servicio determinado.

La siguiente entrada, en el archivo */etc/inittab*, etiquetada como *ca*, se ejecuta cuando se presiona simultáneamente la combinación de las teclas *Ctrl+Alt+Supr*. Esta combinación de teclas proporciona una interrupción que normalmente reinicia el sistema. En Linux, esta interrupción se capta y se envía a *init*, que ejecuta la entrada con el campo *action* de *ctrlaltdel*. Los comandos especificados aquí, */sbin/shutdown -t3 -rf now*, realizarán un reinicio *seguro* del sistema. De este modo protegemos al sistema de un reinicio inesperado cuando presionamos *Ctrl+Alt+Supr*.

Por último, el archivo *inittab* incluye entradas que ejecutan */sbin/mingetty* para las primeras seis consolas virtuales. *mingetty* es una de las diversas variantes de *getty* disponibles para Linux. Estos programas permiten iniciar sesión en terminales; sin ellos, el terminal estaría deshabilitado y no respondería cuando un usuario presionara una tecla o el botón del ratón. Los diversos comandos *getty* abren un dispositivo de terminal, como una consola virtual o una línea serie, establecen diversos parámetros para el controlador del terminal y ejecutan */bin/login* para iniciar una sesión en dicho terminal. Por consiguiente, para permitir inicios de sesión en una determinada consola virtual, se tiene que estar ejecutando *getty* o *mingetty* en ella. *mingetty* es la versión utilizada sobre diversos sistemas Linux, pero otros utilizan *getty* o *agetty*, que tienen una sintaxis ligeramente diferente.

mingetty acepta un argumento: un nombre de dispositivo. El puerto denomina las consolas virtuales de Linux como */dev/tty1*, */dev/tty2*, etc. *mingetty* supone que el nombre de un determinado dispositivo es relativo a */dev*. La tasa de baudios para las consolas virtuales generalmente es de 38.400, razón por la que *mingetty*, al contrario que, por ejemplo, *agetty*, tiene su valor predeterminado en dicha cantidad y no requiere especificarse explícitamente.

Cabe destacar que el campo *action* para cada entrada *mingetty* es *respaw*, lo que significa que *init* debe reiniciar el comando dado en la entrada cuando se cierre el proceso *mingetty*, algo que se produce cada vez que cerramos la sesión.

3. Modo de un solo usuario

Normalmente utilizaremos el sistema en un modo de múltiples usuarios para que los usuarios puedan iniciar la sesión. Pero existe un estado especial denominado modo de un solo usuario en el que se está ejecutando Unix pero no el indicador de comandos de inicio. En ese modo, básicamente el usuario existente es el súper usuario (*root*). Puede que tengamos que utilizar este modo durante la instalación de algo que va mal, como para revisar sistemas de archivos dañados.

En el modo de un solo usuario, el sistema es muy poco útil; hay muy poca configuración, los sistemas de archivos se desmontan, etc. Este modo es necesario para recuperarnos frente a ciertos tipos de problemas. Debemos tener en cuenta que Unix sigue siendo un sistema de múltiples procesos, incluso en modo de un solo usuario. Puede ejecutar múltiples programas simultáneamente. Los servicios se pueden ejecutar en segundo plano para que puedan ejecutarse

funciones especiales, como la red. Pero si nuestro sistema admite más de un terminal, sólo podremos utilizar la consola y no se podrá ejecutar el sistema *X Windows*.

4. Cierre del sistema

Afortunadamente, cerrar un sistema Linux es mucho más simple que arrancarlo e iniciarlo. Sin embargo, no es sólo cuestión de presionar el botón de apagado del ordenador. Linux como todos los sistemas Unix, guarda temporalmente en los buffers de memoria las lecturas y escrituras del disco, lo que significa que las escrituras del disco se retrasan todo lo necesario y se sirven múltiples lecturas en el mismo bloque del disco directamente desde la RAM, incrementando así extraordinariamente el rendimiento, ya que los discos son extremadamente lentos con relación a la CPU.

El problema es que si el sistema tuviese que cerrarse o reiniciarse repentinamente, los buffers en memoria no se escribirían en el disco y se podrían perder o dañar los datos. El núcleo vacía los buffers modificados, es decir, los que han cambiado desde que fueron leídos desde el disco, copiándolos de nuevo al disco cada cinco segundos aproximadamente, dependiendo de la configuración, para evitar un daño serio si el sistema se detiene repentinamente. Sin embargo, para estar completamente seguro, el sistema necesita que se efectúe un *cierre seguro* antes de poderse reiniciar, algo que asegurará no sólo que los buffers del disco se sincronizarán correctamente, sino que además permitirá que todo el proceso de salida se realice limpiamente.

El comando *shutdown* es el comando general utilizado para detener o reiniciar el sistema. Como *root*, podemos emitir el comando:

```
$> sudo shutdown -r +10
```

Con este comando conseguimos que el sistema se reinicie en diez minutos. La opción *-r* indica que el sistema se debe reiniciar tras su cierre y *+10* es la cantidad de tiempo, expresada en minutos, que se debe esperar hasta el cierre. Podemos añadir nuestro propio mensaje de aviso incluyéndolo en la línea de comandos, como en el siguiente ejemplo:

```
$> sudo shutdown -r +10 "Reinicio del sistema para cargar nuevo servicio"
```

También podemos especificar un tiempo absoluto para el cierre, como en el siguiente ejemplo:

```
$> sudo shutdown -r 13:00
```

Esto logrará hacer que el sistema se reinicie a la 1:00 PM. Asimismo, podemos utilizar:

```
$> sudo shutdown -r now
```

Para lograr reiniciar el sistema inmediatamente; tras el proceso de cierre seguro. Si hacemos uso de la opción *-h* en lugar de la opción *-r*, el sistema simplemente se detendrá tras el cierre; y después podremos apagar el equipo sin miedo a perder ningún dato. Si no especificamos la opción *-h* ni *-r*, el sistema entrará en modo de un solo usuario.

Como hemos podido comprobar, podemos hacer que *init* capture la secuencia de teclas *Ctrl+Alt+Supr* y ejecutar un comando *shutdown* en respuesta a dicha combinación de teclas. Si estamos acostumbrados a reiniciar de este modo nuestro sistema, es recomendable comprobar si el archivo */etc/inittab* contiene una entrada *ctrlaltdel*. Debemos tener en cuenta que nunca

debemos reiniciar nuestro sistema Linux pulsando el botón de apagado o el conmutador de reinicio del mismo. A no ser que el sistema se quede suspendido de repente, debemos utilizar siempre el comando *shutdown*. Lo extraordinario de un sistema de múltiples procesos es que un programa se puede quedar suspendido, pero casi siempre podemos cambiar a otra consola virtual para su recuperación.

El comando *shutdown* proporciona otras opciones como la opción *-c* la cual cancelará un cierre que se esté ejecutando actualmente. Evidentemente, otra forma de cancelar el cierre puede ser deteniendo el proceso mediante el comando *kill*, pero *shutdown -c* hace que esta tarea sea mucho más fácil.

TEMA 5: ADMINISTRACIÓN DE USUARIOS, GRUPOS Y PERMISOS

Objetivos

- Destacar la importancia de los elementos operativos para la correcta administración de las cuentas de usuario.
- Estudiar los elementos y términos completos vinculados con la administración de cuentas de usuario.
- Profundizar sobre la metodología de autenticación existente para los sistemas Linux.
- Destacar las características de los comandos útiles a la hora de administrar las cuentas de usuario.

Contenido

1. Tipos de cuentas de usuario
 - 1.1. Cuenta de usuario completa o cuenta *shell*
 - 1.2. Cuenta de acceso restringido
 - 1.3. Programas y *daemons*
2. Usuarios y *uids*
3. Pseudo-conexiones
4. Gestión de usuarios y grupos
 - 4.1. El archivo *passwd*
 - 4.2. Contraseñas ocultas
 - 4.3 El archivo de grupo
 - 4.4 El archivo *gshadow*
5. Operaciones con las cuentas de usuario
 - 5.1. Crear
 - 5.2. Eliminar
 - 5.3. Deshabilitar
 - 5.4. Modificar
6. Otro método de autenticación: *PAM*
 - 6.1. ¿Qué es *PAM*?
 - 6.2. Grupos de gestión
 - 6.3. Arquitectura

Bibliografía

Básica

- M Carling, Stephen Degler, James Dennis, “Administración de Sistemas Linux Guía Avanzada”. Editorial Prentice Hall, 2000.
- Dee-Ann LeBlanc, “Administración de sistemas LINUX La biblia”. Editorial ANAYA MULTIMEDIA, 2001.
- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada “, Editorial Anaya Multimedia, 2006.

Complementaria

- Imobach González Sosa, Manolo Padrón Martínez, “Pluggable Authentication Modules (PAM) Versión 1.0 Curso 2003/2004”, GNU Free Documentation License.
http://sopa.dis.ulpgc.es/ii-aso/portal_aso/leclinux/seguridad/pam/pam_doc.pdf

La administración de cuentas de usuario es fundamental para los administradores de sistemas. No sólo absorbe gran cantidad del tiempo de los mismos (la cantidad de tiempo absorbido depende del número total de usuarios y la frecuencia con la que se añaden o eliminan cuentas), sino que suele ser una de las fuentes más comunes de fracasos. Algunos usuarios no están nunca satisfechos con sus cuentas, ya sea por la *shell*, por la cantidad de espacio de disco duro que se les fue asignado, por el nombre de usuario o por la configuración del escritorio virtual. Los sistemas Linux proporcionan herramientas muy potentes para la administración de usuarios y de grupos.

1. Tipos de cuentas de usuario

Muchas personas piensan que solamente hay un tipo de cuenta de usuario en los sistemas Linux. Esta suposición no sólo es incorrecta sino que conduce a fallos de seguridad que pueden ser evitados fácilmente. Existen, en esencia, tres tipos de cuentas que se utilizan en los sistemas Linux: La **primera** denominada *cuenta de usuario completa* o *cuenta shell*, es el tipo de cuenta de entrada en la que la mayoría de usuarios piensa cuando se trata de un sistema Linux, es la típica cuenta que asocia a un usuario del sistema con una contraseña. La **segunda** es la *cuenta de acceso restringido*, en el que el usuario sólo tiene acceso a servicios específicos de la máquina. Y la **tercera** no es una cuenta para usuarios humanos sino para *daemons* (demonios) y otros programas que necesitan ejecutarse. Ejecutar estos programas en la cuenta *root* es peligroso, porque se les asigna un nivel de acceso demasiado alto; es mejor darles su propia cuenta donde tienen exactamente el acceso y control que necesitan. De esa manera, si esos procesos se ponen en peligro o algo va mal, solamente pueden causar problemas en su propia área.

1.1. Cuenta de usuario completa o cuenta *shell*

Una *cuenta de usuario completa* es la más usual de las cuentas de un sistema Linux. Ofrece un acceso *shell* de entrada al sistema así como el uso de cualquier otro elemento al que un usuario tiene permiso ya sea para ejecutarlo o abrirlo. Este tipo de cuenta se crea con comandos rápidos como *useradd*. Desdichadamente, el acceso *shell* es de algún modo un serio fallo de seguridad. Si alguien tiene acceso a la línea de comandos, que es lo que ofrece una *cuenta shell*, les ahorra un paso para irrumpir en el sistema. Con una cuenta de acceso *shell*, los usuarios pueden compilar programas explícitamente para la configuración de su máquina, ejecutarlos con propósitos malévolos, y explotar el sistema de archivos buscando debilidades en el permiso y en la estructura de posesión. Incluso si la persona que posee la cuenta no tiene intenciones hostiles, introducir una mala contraseña puede poner la cuenta en peligro.

La **regla general**, para proceder como buenos administradores, es que solamente debemos proporcionar *cuentas de usuario completas* si realmente estos necesitan acceso *shell*.

1.2. Cuenta de acceso restringido

Se conocen como *no-shell* y son cuentas que sólo permiten a los usuarios tener acceso a ciertas características específicas: Correo electrónico POP, noticias Usenet o acceso al mercado SLIP o PPP.

Estas cuentas se pueden cambiar a *cuentas shell* en cualquier momento si es necesario. La ventaja de permitir a los usuarios el acceso únicamente a servicios específicos, debería estar clara por los motivos mencionados en el apartado anterior. Si los usuarios solamente tienen acceso para comprobar su correo electrónico POP, no pueden llegar a una línea de comandos para causar estragos en el sistema.

Las contraseñas robustas y seguras son tan importantes con las *cuentas de acceso restringido* como lo son con las *cuentas shell*.

1.3. Programas y *daemons*

Algunos programas y *daemons* necesitan sus propios ID de usuario. Un programa o *daemon* se ejecuta como el usuario que lo posee, así que si el elemento se ejecuta como *root*, tiene todos los privilegios que tiene *root*. Lo hace tan peligroso como un usuario inexperto que inicia una sesión en el sistema como *root*.

Un programa o *daemon* dañino puede estropear accidentalmente un sistema de archivos si sus permisos son demasiado amplios. También es acertado, no dar a los programas acceso de lectura, escritura o ejecución a áreas que no necesitan ver.

2. Usuarios y *uids*

Los *uids* identifican a los propietarios de los archivos, directorios y procesos. Cada *uid* numérico corresponde a un nombre de usuario (o conexión) y opcionalmente a un nombre completo de usuario. La asignación de *uid* a un nombre de usuario se suele encontrar en el archivo */etc/passwd*.

Generalmente, se puede usar un nombre de usuario para conectarse en uno o más sistemas, aunque unos pocos, llamados *pseudo-conexiones*, no pueden conectarse con ningún sistema. Un nombre de usuarios suele pertenecer a una sola persona, y casi todas las personas suelen tener un solo nombre de usuario. *root* es una excepción obvia, ya que en la mayoría de sistemas más de una sola persona utiliza *root*, y todos los que la usan también deben tener un nombre de usuario personal.

Todos los *uid* tienen su contraseña correspondiente, que está almacenada en el archivo */etc/passwd* o en */etc/shadow*. Todos los *uid* también tienen otros atributos, como el *shell* predeterminado.

3. Pseudo-conexiones

Las *pseudo-conexiones* existen para que los procesos puedan ejecutarse con el *uid* de las *pseudo-conexiones* sin que ningún usuario pueda conectarse con ninguna máquina utilizando el correspondiente nombre de usuario. Algunas de las *pseudo-conexiones* que hay en casi todos los sistemas Linux son: *bin*, *daemon*, *adm*, *lp*, *mail*, *news*, *uucp* y *nobody*.

Adicionalmente, es posible crear *pseudo-conexiones* si fuera necesario. Algunos administradores de sistemas las crean con un nombre de usuario que coincide con un comando que tiene que ser ejecutado en su *shell*. Los ejemplos son: *hostname*, *ifconfig*, *netstat* y *who*. Hay que tener en cuenta la seguridad cada vez que se va a crear una *pseudo-conexión*, especialmente si los intentos de usar ese nombre de usuario terminan en la ejecución de un comando. Es importante documentar la razón de la creación de cada *pseudo-conexión*.

4. Gestión de usuarios y grupos

En esta sección analizaremos el contenido y la lógica de funcionamiento de los archivos relacionados con la gestión de usuarios (*/etc/passwd* y */etc/shadow*) y grupos (*/etc/group* y */etc/gshadow*) en un sistema Linux.

4.1. El archivo *passwd*

Todas las cuentas en el sistema tienen una entrada en el archivo */etc/passwd*. Este archivo contiene entradas, una línea por usuario, que especifican diversos atributos para cada cuenta, como el nombre de conexión del usuario, el nombre real del usuario, etc.

Todas las entradas en este archivo tienen el siguiente formato:

```
nombre-de-usuario:contraseña:uid:gid:gecos:homedir:shell
```

La siguiente lista explica cada uno de los campos:

- **nombre-de-usuario:** Es una cadena de caracteres única que identifica la cuenta. En las cuentas personales, es el nombre del usuario que inicia la sesión. En la mayoría de sistemas se limita a ocho caracteres alfanuméricos (por ejemplo, *larry* o *kirsten*).
- **contraseña:** Es una representación cifrada con *MD5* de la contraseña del usuario. Este campo se establece utilizando el comando *passwd* para establecer la contraseña de la cuenta; utiliza un esquema de cifrado de una sola vía, que es difícil, aunque no imposible, de descifrar. Esta contraseña no debe establecerse manualmente; el comando *passwd* lo hace por nosotros. Sin embargo, si el primer carácter del campo contraseña es un * (asterisco), la cuenta estará “deshabilitada”; el sistema no permitirá iniciar ninguna sesión a este usuario. En cambio, si el campo contraseña contiene sólo una *x* entonces quiere decir que la contraseña se encuentra cifrada en el archivo */etc/shadow* el cual es sólo legible por *root*.
- **uid:** Es el ID del usuario, un entero único que utiliza el sistema para identificar la cuenta. El sistema utiliza el campo *uid* internamente cuando trata con procesos y permisos de archivos; es más fácil y más compacto para el sistema tratar con enteros que con cadenas de bytes. Por consiguiente, tanto el ID del usuario como el nombre del usuario identifican una determinada cuenta: El ID del usuario es más importante para el sistema, mientras que el nombre del usuario es más cómodo para éste.
- **gid:** Es el ID de grupo, un entero que hace referencia al grupo predeterminado del usuario, que se encuentra en el archivo */etc/group*.
- **gecos:** Es información diversa sobre el usuario, como su nombre real, información de localización, dirección de su oficina o su número de teléfono. Comandos como *mail* y *finger* utilizan esta información para identificar a los usuarios del sistema. *gecos* es un nombre histórico que data del año 1970 y es el acrónimo de General Electric Comprehensive Operating System usado para mandar *login* en trabajos *batch* de Unix a mainframe Bell. Podemos hacer uso del comando *chfn*, para modificar este campo de un usuario existente en nuestro sistema.
- **homedir:** Es el directorio personal del usuario. Cuando los usuarios inician la sesión por primera vez, la *shell* busca su directorio de trabajo actual en el denominado directorio personal.
- **shell:** Es el nombre del programa a ejecutar cuando el usuario inicia la sesión; normalmente es el nombre de la ruta de acceso completo de una *shell*, como */bin/bash* o */bin/sh*. Podemos hacer uso del comando *chsh*, para modificar este campo de un usuario existente en nuestro sistema.

Muchos de estos campos son opcionales; los únicos campos obligatorios son *nombre-de-usuario*, *uid*, *gid* y *homedir*. La mayoría de cuentas de usuario tienen todos los campos rellenos,

pero las cuentas “imaginarias” o administrativas pueden utilizar sólo algunos de ellos.

Éstos son dos ejemplos de entradas que se puede encontrar en */etc/passwd*:

```
root:ZxPsI9ZjiVd9y:0:0The root of all evil:/root:/bin/bash
aclark:BjDf5hBysDsii:1000:1000:Anna Clark:/home/aclark:/bin/bash
```

La primera entrada es para la cuenta *root* (el ID del usuario *root* es 0). Lo que hace que *root* sea *root* es que el sistema sabe que un *uid* 0 es *especial* y que no tiene que tener las restricciones de seguridad normales. El *gid* de *root* también es 0 (principalmente se trata de una comodidad). Muchos de los archivos del sistema son propiedad de *root* y del grupo *root*, que tiene un *uid* y un *gid* de 0, respectivamente.

En muchos sistemas, *root* utiliza el directorio personal */root* o simplemente */*. Normalmente no es algo importante ya que por lo general utilizaremos el comando *su* para acceder a *root* desde nuestra propia cuenta. Asimismo, es tradicional utilizar una variante del *shell Bourne* (intérprete de comandos y lenguaje de programación de comandos), en este caso, */bin/bash*, para la cuenta *root*.

La segunda entrada es para la identificación de una persona, con nombre de usuario *aclark* y *uid* 1000. Técnicamente el campo *uid* puede ser cualquier entero único dentro del sistema superior o igual a 1000, ya que las cuentas administrativas van de 0 hasta 999; el *gid* es 1000, lo que significa que *aclark* se encuentra en el grupo con el número 1000 del archivo */etc/group*.

Los directorios personales normalmente se encuentran en */home* y se denominan por el nombre de usuario de su propietario, por ejemplo: */home/aclark*, el cual es un convenio útil que evita confusiones cuando se busca el directorio personal de un determinado usuario. Técnicamente se puede colocar un directorio personal en cualquier parte del sistema, pero primero tiene que existir dicho directorio para que el usuario pueda iniciar su sesión. Sin embargo, como buenos administradores, debemos seguir el diseño del directorio utilizado en nuestro sistema.

Como administrador del sistema, en general, no es necesario modificar el directorio */etc/passwd* directamente. Existen diversos comandos disponibles que pueden ayudarnos a crear y a mantener cuentas de usuario. Si realmente deseamos modificar directamente los datos del archivo */etc/passwd*, podemos considerar utilizar el comando *vipw* que protege el archivo de contraseñas ante los potenciales daños que se pueden producir de una edición simultánea.

4.2. Contraseñas ocultas

Hasta cierto punto, es riesgo de seguridad permitir que todo el que tenga acceso al sistema pueda ver las contraseñas cifradas en el archivo */etc/passwd*. Existen comandos especiales para descifrar contraseñas que intentan una inmensa cantidad de contraseñas posibles y comprueban si la versión cifrada de dichas contraseñas es igual a una especificada.

Para solucionar en parte este potencial riesgo de seguridad, se han desarrollado las contraseñas ocultas (*shadow passwords*). Cuando se utilizan las contraseñas ocultas, el campo contraseña en */etc/passwd* contiene sólo una *x*, que nunca pueden estar en la versión cifrada de una contraseña. En su lugar, se utiliza un segundo archivo denominado */etc/shadow*, sólo legible por *root*, por lo que los usuarios normales no tienen acceso a las contraseñas cifradas. Los otros campos de */etc/shadow*, excepto el nombre de usuario y la contraseña, también están presentes, pero normalmente contienen valores falsos o están vacíos, para tratar de confundir a un posible atacante a la hora de que este pueda leer su contenido.

Todas las entradas en este archivo tienen el siguiente formato:

nombre-de-usuario:contraseña:días-cambio:min-cambio:max-cambio:días-aviso:días-inhabilitar:tiempo-inhabilitar:reservada

La siguiente lista explica cada uno de los campos:

- **nombre-de-usuario:** Igual que el campo de */etc/passwd*.
- **contraseña:** Es una representación cifrada con *MD5* de la contraseña del usuario.
- **días-cambio:** Define la fecha de nacimiento, o de la última modificación, de la contraseña respecto al 1 de enero de 1970 “Unix timestamp” expresada en días.
- **min-cambio:** Número entero que establece la edad mínima en días que tiene que tener la contraseña para que se pueda cambiar, es decir, los días que faltan para que se cambie la contraseña. Un cero indica que el usuario puede cambiar la contraseña en cualquier momento. Es un campo opcional.
- **max-cambio:** Número entero que establece la edad máxima en días de la contraseña. Establece el vencimiento de la contraseña. Es un campo opcional.
- **días-aviso:** Numero entero que establece los días antes de la edad máxima (*max-cambio*), que el sistema comenzará a solicitar al usuario que cambie la contraseña.
- **días-inhabilitar:** Plazo en días que se concede si se caduca la contraseña sin cambiar para que el sistema inhabilite la cuenta del usuario. Por ejemplo: 2, al cabo de dos días de expirar la contraseña se inhabilitará la cuenta.
- **tiempo-inhabilitar:** Define el número de días después del cual se inhabilitará la cuenta; y el usuario no podrá iniciar su sesión en el sistema. Es un campo opcional.
- **reservada:** Campo reservado.

Para utilizar las contraseñas ocultas se necesitan versiones especiales de comandos que accedan o modifiquen la información del usuario, como *passwd* o *login*. Actualmente, la mayoría de distribuciones Linux incluyen una configuración de contraseña oculta.

Existen dos comandos para convertir las entradas de usuario “normales” en entradas ocultas y viceversa. Por ejemplo, *pwconv* busca en el archivo */etc/passwd* entradas que todavía no estén en */etc/shadow*, genera entradas ocultas para dichas entradas y las combina con las entradas ya presentes en */etc/shadow*.

El comando *pwunconv* se utiliza en raras ocasiones ya que proporciona menos seguridad en lugar de más. Funciona como *pwconv*, pero genera entradas de */etc/passwd* tradicionales que funcionan sin sus equivalentes en */etc/shadow*.

Los sistemas Linux modernos también proporcionan una utilidad denominada caducidad de la contraseña (*password aging*). Se trata de una fecha de caducidad de una contraseña; si llega a esta fecha, se emite un aviso, unos cuantos días antes (dichos días pueden ser configurados por el administrador) de que la contraseña caduque y se le pide al usuario que cambie su contraseña. Si éste no lo hace, su cuenta se bloqueará. También se puede establecer una cantidad mínima de días antes de que se cambie o se cree una contraseña para que se pueda volver a cambiar.

Todas estas configuraciones se establecen con el comando *passwd*. La opción *-n* establece la cantidad mínima de días entre cambios, *-x* la cantidad máxima de días entre cambios, *-w* cuántos

días antes se debe emitir un aviso antes de que caduque la contraseña y *-i* la cantidad de días de inactividad entre la fecha de caducidad de una contraseña y el momento en el que se bloquea la cuenta.

4.3. El archivo de grupo

Los grupos proporcionan un mecanismo para agregar usuarios, de forma que se pueden garantizar permisos repetidamente a un grupo de usuarios sin la necesidad de enumerar los miembros cada vez. Además es un método conveniente de organizar conjuntos de cuentas de usuarios de forma lógica y permitir que los usuarios compartan archivos dentro de su grupo o grupos. La información acerca de los grupos a los que pertenece un usuario se deriva del archivo */etc/group*.

Cada archivo del sistema tiene tanto un usuario propietario como un grupo propietario asociado a él. Con el comando *ls -l* (ver figura 5.1) podemos ver la propiedad y el grupo de un determinado archivo, como en el siguiente ejemplo:

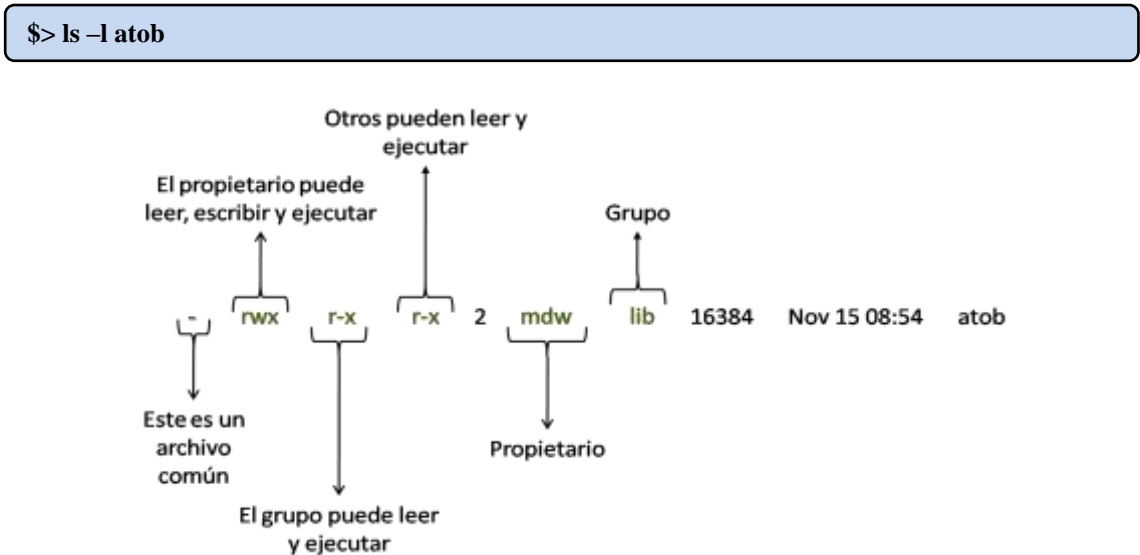


Figura 5.1: Propiedad y grupo de un archivo

Este archivo es propiedad del usuario *mdw* y pertenece al grupo *lib*. Podemos comprobar a partir de los permisos del archivo que *mdw* tiene autorización de acceso de lectura, escritura y ejecución sobre dicho archivo, que cualquier miembro del grupo *lib* tiene acceso de lectura y ejecución y que el resto de usuarios también tienen acceso de lectura y ejecución.

Lo anterior no significa que *mdw* se encuentre dentro del grupo *lib*; simplemente significa que, tal como muestran los bits de permisos, cualquiera que pertenezca al grupo *lib* (que puede incluir o no a *mdw*) puede acceder al archivo. De este modo, se pueden compartir archivos entre grupos de usuarios y se pueden especificar permisos de forma independiente para el propietario del archivo, el grupo al que pertenece dicho archivo y cualquier otro usuario.

Cada usuario está asignado, al menos, a un grupo, que podemos especificar en el campo *gid* del archivo */etc/passwd*. No obstante, un usuario puede ser miembro de múltiples grupos. El archivo */etc/group* contiene una entrada de una línea para cada grupo del sistema, muy similar en naturaleza al archivo */etc/passwd*.

Todas las entradas en este archivo tienen el siguiente formato:

nombre-de-grupo:contraseña:gid:miembros

La siguiente lista explica cada uno de los campos:

- **nombre-de-grupo:** Es una cadena de caracteres que identifica al grupo; es el nombre del grupo que se imprime cuando utilizamos comandos como *ls -l*.
- **contraseña:** Es una contraseña cifrada opcional asociada al grupo pero, si hacemos uso de este campo aumentamos la seguridad de acceso a los recursos del grupo. Permite a usuarios no incluidos en el grupo acceder al mismo con el comando *newgrp*. Si el valor del campo es sólo una *x* indica que la contraseña cifrada esta en el archivo */etc/gshadow* sólo legible por *root* y por otro lado, si el valor del campo esta en blanco indica que el grupo no tiene contraseña.
- **gid:** Es el ID del grupo, utilizado por el sistema para hacer referencia al grupo; se trata del número utilizado en el campo *gid* del archivo */etc/passwd* para especificar el grupo predeterminado de un usuario.
- **miembros:** Es una lista de nombres de usuarios separados por comas, sin espacios en blanco entre ellos, que identifican a los usuarios miembros de este grupo pero que tienen un *gid* diferente en */etc/passwd*. Es decir, esta lista no contiene necesariamente a todos los usuarios que tienen establecido a éste como grupo *predeterminado* en */etc/passwd*; es sólo para usuarios que son miembros adicionales del grupo.

Por ejemplo, */etc/group* puede contener las siguientes entradas:

```
root:x:0
bin:x:2
users:x:100:
bozo:x:1000:linus,mdw
megaboza:x:1001:kibo
```

Las primeras entradas, para los grupos *root* y *bin*, indican que son grupos administrativos, similares en naturaleza a las cuentas “imaginarias” utilizadas en el sistema. Muchos archivos son propiedad de los grupos, como *root* y *bin*. El resto de grupos son para cuentas de usuario. Igual que los ID de usuario, los valores de ID de los grupos normalmente se colocan en rangos por encima o iguales a 1000 (Para los grupos especiales (administrativos) los valores de los ID serán menores o iguales a 999).

El campo *contraseña* del archivo */etc/group* es muy curioso. No se utiliza mucho, pero junto con el comando *newgrp*, permite a usuarios que no son miembros de un determinado grupo asumir ese ID de grupo si disponen de contraseña.

Por ejemplo:

```
$> newgrp bozo
Password: contraseña para el grupo bozo
```

Sin embargo, raras veces se utiliza el campo de contraseña de un archivo de grupo. De hecho, la mayoría de sistemas no proporcionan herramientas para establecer la contraseña para un grupo; lo que podemos hacer es utilizar *passwd* para establecer la contraseña para un usuario con el mismo nombre del grupo en */etc/passwd* y copiar el campo de la contraseña cifrada en */etc/gshadow*. Otra opción es hacer que un usuario sea miembro de múltiples grupos simplemente incluyendo su nombre de usuario en el campo *miembros* para cada grupo adicional. En el ejemplo anterior, los usuarios *linus* y *mdw* son miembros del grupo *bozo*, así

como de cualquier grupo al que estén asignados en el archivo */etc/passwd*. Si quisiéramos añadir *linus* al grupo *megaboza*, tendríamos que cambiar la última línea del ejemplo anterior por la siguiente:

```
megaboza:x:1001:kibo,linus
```

El comando *groups* nos indica a qué grupo o grupos pertenecemos:

```
$> groups  
users bozo
```

Si proporcionamos una lista de nombres de usuarios a este comando, obtendremos una lista de los grupos a los que pertenece cada uno de los usuarios especificados en la lista.

Cuando un usuario inicia la sesión, se le proporciona automáticamente el ID de grupo proporcionado en */etc/passwd*, así como a cualquier grupo adicional al que se encuentre asignado dicho usuario en */etc/group*. Es decir, tendrá “acceso de grupo” a cualquier archivo del sistema que tenga un ID de grupo contenido en su lista de grupos. En este caso, se le aplicarán a sus archivos los bits de permisos del grupo (establecidos con *chmod +g...*), a no ser que sea el propietario de los archivos, en cuyo caso se le aplicarán en su lugar los permisos de propietario.

4.4. El archivo *gshadow*

Es la correspondencia del archivo */etc/shadow* pero aplicado a nivel de grupos. Es un fichero de texto, donde cada línea tiene información de un grupo definido en */etc/group*. Nos permite guardar las contraseñas cifradas de los grupos y es sólo visible por *root*.

Todas las entradas en este archivo tienen el siguiente formato:

```
nombre-de-grupo:contraseña:administradores:miembros
```

La siguiente lista explica cada uno de los campos:

- **nombre-de-grupo:** Nombre del grupo, igual que en */etc/group*.
- **contraseña:** Contraseña cifrada para el grupo. Si el valor de este campo es *!*, entonces ningún usuario tiene acceso al grupo usando el comando *newgrp*; por otro lado si el valor de este campo es *** (asterisco), ningún usuario podrá usar una contraseña para iniciar sesión.
- **administradores:** Es una lista delimitada por comas de nombres de usuarios que son administradores del grupo y pueden añadir o eliminar usuarios (*miembros*) al grupo usando el comando *passwd*.
- **miembros:** Es una lista delimitada por comas de nombres de usuarios que son miembros del grupo.

La idea de introducir las contraseñas de grupo dentro de otro archivo (*/etc/gshadow*), es la misma idea de introducir las contraseñas de usuario dentro otro archivo (*/etc/shadow*). Ya que en ambos casos el problema radica en que tanto */etc/passwd* como */etc/group* son archivos legibles por cualquier usuario, con lo que un atacante podría visualizar el contenido del archivo y efectuar un ataque de fuerza bruta, hasta que descriptase las contraseñas contenidas en dichos archivos. Ahora tanto */etc/shadow* como */etc/gshadow* son los archivos que contienen la contraseña de usuario y de grupo respectivamente, y todo esto es únicamente visible por *root*.

5. Operaciones con las cuentas de usuario

En la siguiente sección haremos una breve descripción de cuales son algunas de las operaciones que podemos realizar sobre las cuentas dentro de un sistema operativo Linux.

5.1. Crear

La creación de una cuenta de usuario requiere el seguimiento de diversos pasos: Añadir una cuenta en el archivo `/etc/passwd` e introducir su contraseña cifrada en el archivo `/etc/shadow`, crear el grupo primario de la cuenta en los archivos `/etc/group` y `/etc/gshadow`, crear el directorio personal de la cuenta en `/home`, y establecer los archivos de configuración predeterminados de la cuenta (como `.bashrc`) en su directorio personal (ver figura 5.2). Afortunadamente, no tenemos que ejecutar manualmente estos pasos (aunque si quisiéramos, lo podríamos hacer); casi todos los sistemas Linux, por no decir todos, incluyen un comando denominado `adduser` para ejecutar esta tarea.

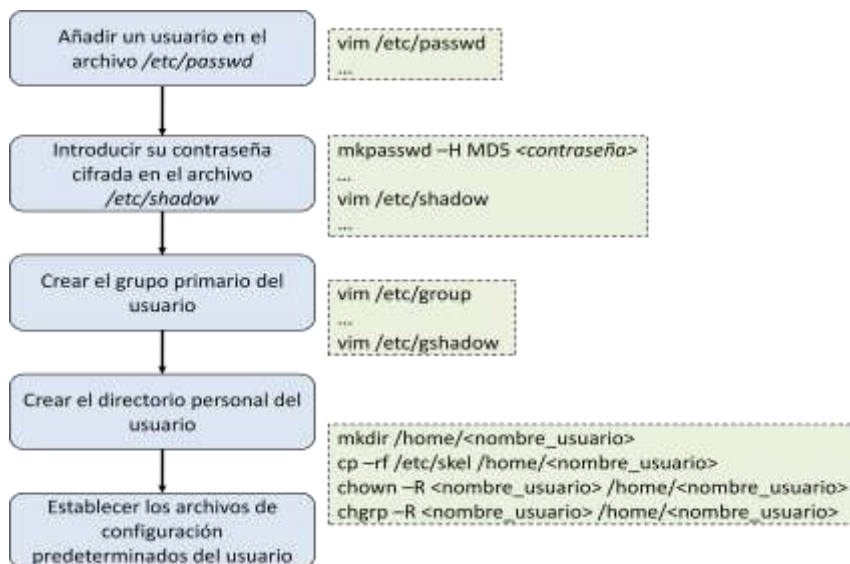


Figura 5.2: Esquema típico de creación de un usuario

La ejecución de `adduser` como `root` funciona como sigue. Sólo tenemos que pasar como argumento de la línea de comandos el nuevo usuario que queremos agregar al sistema; y cuando el comando nos la solicite añadiremos la contraseña para dicho usuario, y una que otra información adicional.

Es recomendable que el nombre del usuario no exceda los ocho caracteres de longitud para evitar tener posibles problemas posteriormente.

```

$> sudo adduser pepe
Añadiendo usuario 'pepe' ...
Agregando nuevo grupo `pepe' (1001) ...
Agregando nuevo usuario `pepe' (1001) con grupo `pepe' ...
Creando el directorio personal '/home/pepe' ...
Copiando archivos desde '/etc/skel' ...
Enter new UNIX password: <Contraseña de pepe>
Retype new UNIX password: <Repetir la contraseña de pepe>
passwd: contraseña actualizada correctamente
Cambiando la información de usuario para pepe
Introduzca el nuevo valor, o presione ENTER para el predeterminado
Nombre completo []: Jose Benito Moraga Mendoza
Número de habitación []: ENTER
Teléfono del trabajo []: 311-2626
Teléfono de casa []: 315-8976
Otro []: ENTER
¿Es correcta la información? [y/N] y

```

`adduser` busca el primer ID de usuario e ID de grupo sin utilizar que sea mayor o igual a 1000.

Una vez creada la cuenta, se copian los archivos de `/etc/skel` al directorio personal de usuario. En dichos archivos se encuentra el “esqueleto” de la nueva cuenta; se trata de los archivos de configuración predeterminados (como `.emacs` y `.bashrc`) para el nuevo usuario. Aquí tenemos plena libertad para colocar archivos, si nuestras nuevas cuentas de usuario los necesitan.

Al ejecutar el comando anterior (`adduser`), la cuenta nueva estará preparada: `pepe` puede iniciar una sesión utilizando la contraseña establecida con `adduser`. Para garantizar la seguridad, ahora los nuevos usuarios tienen que modificar sus propias contraseñas utilizando el comando `passwd`, inmediatamente después de iniciar la sesión por primera vez. Para evitar delegar esta tarea en un usuario normal, podemos hacer uso del comando `chage` con la opción `-d 0` lo cual nos va a permitir que cuando un usuario, en este caso `pepe`, inicie sesión por primera vez se le fuerce automáticamente a cambiar su contraseña evitando que dicho usuario haga caso omiso a la tarea que habíamos delegado en él.

Por ejemplo, con el usuario `pepe`:

```
$> sudo chage -d 0 pepe
```

Una vez que el usuario `pepe` quiera ingresar al sistema le aparecerá lo siguiente:

```

$> su pepe
Password: <Contraseña de pepe>
You are required to change your password immediately (root enforced)
Changing password for pepe
(current) UNIX password: <Contraseña actual de pepe>
Enter new UNIX password: <Nueva contraseña para pepe elegida por pepe>
Retype new UNIX password: <Repetir la nueva contraseña para pepe elegida por pepe>
$pepe>

```

La cuenta *root* puede establecer la contraseña de cualquier usuario del sistema. Por ejemplo, el comando: `passwd pepe` solicita una nueva contraseña para *pepe*, sin pedir la contraseña original. Sin embargo, para poder hacer eso tenemos que conocer la contraseña de *root*. Si se nos olvida la contraseña de *root* (algo que jamás tiene que sucedernos como buenos administradores), podemos iniciar Linux desde un disco de emergencia y borrar el campo *contraseña* de la entrada `/etc/passwd` y `/etc/shadow` para *root*.

Por ejemplo:

```
$> sudo passwd pepe
Password: <Contraseña de root>
Enter new UNIX password: <Nueva contraseña para pepe establecida por root>
Retype new UNIX password: <Repetir la nueva contraseña para pepe establecida por root>
passwd: contraseña actualizada correctamente
```

Algunos sistemas Linux proporcionan un comando *useradd* controlador por la línea de comandos en lugar de *adduser*. (Y para complicar aún más las cosas, en otros sistemas, los dos comandos son sinónimos). Este comando requiere que le proporcionemos toda la información relevante como argumentos de la línea de órdenes.

5.2. Eliminar

Eliminar una cuenta de usuario es mucho más fácil que crearla. Para ello, tenemos que hacer lo siguiente: Eliminar la entrada del usuario en `/etc/passwd` y en `/etc/shadow`, suprimir cualquier referencia al usuario en `/etc/group` y en `/etc/gshadow`. Y eliminar el directorio personal del mismo, así como cualquier archivo adicional creado por el usuario o que sea propiedad del mismo; Por ejemplo, si el usuario tiene una cuenta de correo electrónico en `/var/spool/mail`, está tendrá que ser eliminada.

El comando *userdel* (el *yin* para el *yan* *useradd*) elimina una cuenta en el directorio personal de la cuenta. Por ejemplo: `userdel -r aclark` eliminará la cuenta creada para el usuario *aclark*. La opción `-r` obliga también a la eliminación del directorio personal. Otros archivos asociados al usuario (por ejemplo, la cuenta de correo electrónico, los archivos *crontab*, etc.) tienen que ser eliminados manualmente. Un método sencillo para buscar los archivos asociados con un determinado usuario es utilizar el siguiente comando:

```
$> sudo find / -user <nombre_de_usuario> -ls
```

Así obtenemos una lista de todos los archivos que son propiedad de *nombre_de_usuario*. Evidentemente, para utilizar este comando, la cuenta asociada al *nombre_de_usuario* tiene que seguir teniendo una entrada en `/etc/passwd`, es decir, debemos ejecutar este comando antes de borrar, con el comando *userdel*, al usuario del sistema. Si hemos eliminado la cuenta, podemos utilizar en lugar de la opción `-user` la opción `-uid num`, siendo *num* el ID numérico del usuario cuyos archivos queremos eliminar.

Por ejemplo:

```
$> sudo find / -uid <uid_del_usuario> -ls
```

5.3. Deshabilitar

Deshabilitar temporalmente (o no tanto) una cuenta de usuario por cualquier razón, es incluso más fácil. Lo podemos hacer ya sea eliminando la entrada del usuario en `/etc/passwd` y

en */etc/shadow* (sin tocar el directorio personal y otros archivos), o sustituyendo la *x* del campo *contraseña* dentro de */etc/passwd* por un asterisco, si hacemos uso de contraseñas ocultas, de lo contrario solo bastará con añadir un asterisco como primer carácter en el campo *contraseña* de */etc/passwd*.

Por ejemplo:

Con contraseñas ocultas:

```
aclark:*:1000:1000:Anna Clark:/home/aclark:/bin/bash
```

Sin contraseñas ocultas:

```
aclark:*BjDf5hBysDsii:1000:1000:Anna Clark:/home/aclark:/bin/bash
```

Al hacer esto no permitiremos que la cuenta *aclark* inicie sesión. Pero, ¿por qué no debemos tocar su directorio personal y sus otros archivos? Imaginemos que un usuario ha dejado la “empresa” y queremos evitar que siga iniciando la sesión, pero deseamos seguir manteniendo sus archivos durante algún tiempo por si algún otro usuario los llegara a necesitar. En este caso, es conveniente poder deshabilitar la cuenta sin eliminar realmente el directorio personal del usuario (y otros archivos relacionados con la memoria del correo).

La **regla general**, como buenos administradores, es que cuando se prevea que una cuenta de usuario no va a utilizarse, pero no interesa borrarla por cualquier circunstancia, lo mejor que podemos hacer es bloquearla para evitar que sea utilizada indebidamente y pueda comprometer otras cuentas o el sistema.

Existe un mecanismo aún más sencillo para deshabilitar y habilitar las cuentas de usuario de nuestro sistema. Es a través del comando *usermod*; con la opción *-L* bloqueamos una cuenta de usuario.

Por ejemplo:

```
$> sudo usermod -L aclark
```

Esto hará que la cuenta *pepe* permanezca bloqueada en el sistema. Para desbloquearla hacemos nuevamente uso del comando *usermod* pero ahora con la opción *-U*.

Por ejemplo:

```
$> sudo usermod -U aclark
```

5.4. Modificar

Modificar los atributos de las cuentas y los grupos de usuarios es tan sencillo como editar */etc/passwd* y */etc/group*.

Para cambiar la contraseña de un usuario, utilizamos el comando *passwd* que nos pedirá la contraseña actual y luego la nueva contraseña que queremos establecer, el se encargará automáticamente de cifrar y almacenar dicha contraseña en el archivo */etc/shadow*.

Si necesitáramos cambiar el ID de usuario de una cuenta existente, podemos hacerlo directamente modificando el campo *uid* de */etc/passwd*. Sin embargo, también tenemos que

utilizar el comando *chmod* en los archivos propiedad del usuario para ese ID de usuario nuevo.

Por ejemplo:

```
$> chown -R aclark /home/aclark
```

Esto establecerá la propiedad de todos los archivos en el directorio personal utilizado por *aclark* de nuevo para *aclark*.

6. Otro método de autenticación: PAM

Puede que pensemos que tener dos medios de autenticación para usuarios y grupos, */etc/passwd*, */etc/shadow* y */etc/group*, */etc/gshadow*, son más que suficientes para nuestro sistema, pero estamos equivocados. Pues existen otros métodos de autenticación. Aunque creamos que las contraseñas ocultas proporcionan suficiente seguridad en la mayoría de ocasiones, dependerá de cuánta seguridad consideremos que necesita realmente nuestro sistema y de lo persistentes, que como administradores, podamos llegar a ser.

Los *Pluggable Authentication Modules (PAM)* se han convertido en el estándar *de facto* para la autenticación de usuarios en los sistemas Unix. Su gran flexibilidad ofrece a administradores y desarrolladores un control muy valioso.

Gracias a *PAM*, los administradores de sistemas pueden modelar e implementar diferentes políticas de autenticación para los distintos usuarios de forma individualizada para cada servicio. Pero hay que manejar estas facilidades con sumo cuidado, ya que una mala decisión o, simplemente, un despiste pueden comprometer gravemente la seguridad del sistema. Por tanto, el administrador tiene que conocer muy bien cómo funciona *PAM* si realmente quiere afinar al máximo el proceso de autenticación del sistema.

6.1. ¿Qué es PAM?

La idea original de los *Pluggable Authentication Modules (PAM)*, fue de Sun Microsystems. Sin embargo, muchos otros sistemas adoptaron esta solución y cuentan desde hace tiempo con sus propias implementaciones. En este sentido, GNU/Linux no es una excepción y, gracias a Red Hat, disfruta ya desde hace años de la funcionalidad que ofrece *Linux-PAM*.

Pero, ¿qué es PAM exactamente? es, básicamente, un mecanismo flexible para la autenticación de usuarios. Y quizás esta característica, la flexibilidad, sea su aportación más importante.

A lo largo de los años, desde los primeros sistemas Unix, los mecanismos de autenticación han ido evolucionando y han aparecido nuevas opciones: Desde mejoras del */etc/passwd* (como *shadow*) hasta dispositivos hardware orientados a la autenticación. Y cada vez que aparecía y se popularizaba un nuevo método de autenticación, los desarrolladores debían modificar sus programas para darles soporte.

PAM permite el desarrollo de programas independientes del mecanismo de autenticación a utilizar. Así es posible que un programa que aproveche las facilidades ofrecidas por *PAM* sea capaz de utilizar desde el sencillo */etc/passwd* hasta dispositivos hardware (como lectores de huella digital), pasando por servidores *LDAP* o sistemas de gestión de bases de datos. Y, por supuesto, todo esto sin cambiar ni una sola línea de código.

Pero *PAM* va más allá todavía, permitiendo al administrador del sistema construir políticas diferentes de autenticación para cada servicio.

Podemos sintetizar las ventajas más importantes de *PAM* en los siguientes puntos:

- Ofrece un esquema de autenticación común y centralizado.
- Permite a los desarrolladores abstraerse de las labores de autenticación.
- Facilita el mantenimiento de las aplicaciones.
- Ofrece flexibilidad y control tanto para el desarrollador como para el administrador de sistema.

En síntesis, los módulos *PAM* son un método que permite al administrador controlar cómo se realiza el proceso de autenticación de los usuarios para determinadas aplicaciones. Las aplicaciones tienen que haber sido creadas y enlazadas a las bibliotecas *PAM*. Básicamente, los módulos *PAM* son un conjunto de bibliotecas compartidas que pueden incorporarse a las aplicaciones como método para controlar la autenticación de sus usuarios. Es más, puede cambiarse el método de autenticación (mediante la configuración de los módulos *PAM*), sin que sea necesario cambiar la aplicación.

6.2. Grupos de gestión

La misión de *PAM* no es, únicamente, comprobar que un usuario es quien dice ser (autenticación). Su alcance es mucho mayor y sus tareas pueden dividirse en cuatro grupos independientes de gestión, cada uno de los cuales se encarga de un aspecto diferente de los servicios restringidos.

account (cuenta): En este grupo se engloban tareas que no están relacionadas directamente con la autenticación. Algunos ejemplos son permitir o denegar el acceso en función de la hora, los recursos disponibles o incluso la localización. Ofrece verificación de cuentas de usuario. Por ejemplo, se encarga de determinar si el usuario tiene o no acceso al servicio, si su contraseña ha caducado, etc.

authentication (autenticación): Tareas encaminadas a comprobar que, efectivamente, el usuario es realmente quien dice ser. Estas tareas ofrecen incluso un sistema de credenciales que permiten al usuario ganar ciertos privilegios (fijados por el administrador).

password (contraseña): Se encarga de mantener actualizado el elemento de autenticación asociado a cada usuario (por ejemplo, su contraseña). Acciones como comprobar la fortaleza de una clave son típicas de este grupo.

session (sesión): En este grupo se engloban tareas que se deben llevar a cabo antes de iniciarse el servicio y después de que este finalice. Es especialmente útil para mantener registros de acceso o hacer accesible el directorio home del usuario.

6.3. Arquitectura

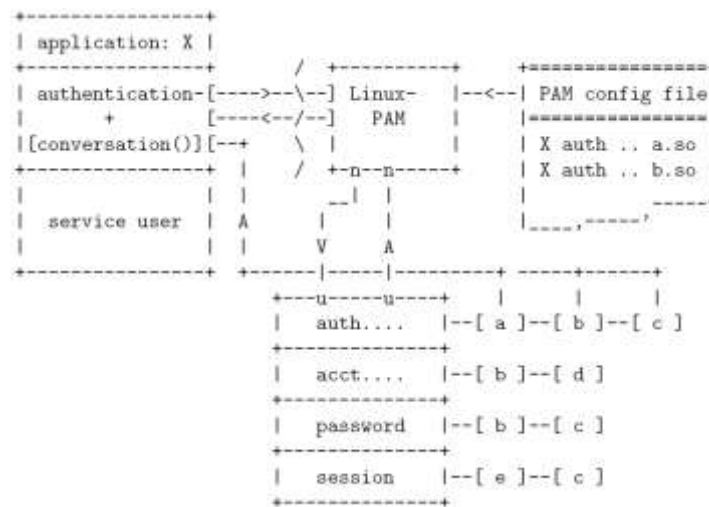


Figura 5.3: Arquitectura de *Linux-PAM*

Dada la figura 5.3, supongamos que la aplicación X quiere hacer uso de las facilidades ofrecidas por PAM. Para ello, interactúa con la biblioteca de Linux-PAM, sin tener que conocer ningún detalle acerca de cómo está configurado el sistema para la aplicación X. Será precisamente esta biblioteca quien se encargue de leer la configuración de PAM para conocer qué política de autenticación ha de aplicarse (combinando de forma conveniente una serie de módulos).

Los módulos se colocan en una pila según el grupo de gestión y el orden en el que aparecen en la configuración (un módulo puede pertenecer a varios grupos), para ser utilizados por PAM cuando corresponda. Este aspecto es tremendamente importante, ya que el orden de los módulos en la pila va a determinar, en gran medida, el comportamiento de PAM para un servicio dado. En la figura 5.3, para la tarea de autenticación, se invocará primero al módulo a, luego al módulo b y, finalmente, al módulo c.

Finalmente, PAM ofrece a la aplicación una serie de funciones para llevar a cabo las diferentes tareas de cada grupo (autenticar, abrir sesión, etc.), mientras que la aplicación brinda a PAM una función de conversación destinada a intercambiar información textual. Gracias a esta función, PAM se libera de tener que preocuparse de cómo enviar/recibir información del cliente (cuadros de diálogo, intercambio en un terminal, protocolos de red, etc.).

TEMA 6: EL SISTEMA DE ARCHIVOS

Objetivos

- Promover los conocimientos y hábitos para la aplicación libre de errores, de términos y elementos propios de un sistema de archivos Linux.
- Comprender la estructura del sistema de archivos de Linux.
- Administrar correctamente los procedimientos sobre los permisos de archivos.

Contenido

1. Concepto de archivo y de sistema de archivos
2. Los *inodos*
3. El superbloque
4. El sistema de archivos *ext2*
5. El sistema de archivos *ext3*
 - 5.1. ¿Cómo surge *ext3*?
 - 5.2. ¿Qué es *journaling*?
6. El estándar de jerarquía del sistema de archivos
7. Algunos directorios interesantes
8. Nombres de archivos y directorios
 - 8.1 Convenios en los nombres de los archivos
9. Tipos de archivos
 - 9.1. Archivos normales
 - 9.2. Archivos de directorio
 - 9.3. Directorios y discos físicos
 - 9.4. Enlaces
 - 9.5. Archivos especiales
10. Atributos existentes en el sistema de archivos de Linux
11. Propiedad y permisos de los archivos
 - 11.1. ¿Qué significan los permisos?
 - 11.2. Propietarios y grupos
12. Puntos adicionales del sistema de archivos
 - 12.1. Máscara frente a umáscara
 - 12.2. Establecer ID de usuario y de grupo (*SUID* y *SGID*)
 - 12.3. El Sticky bit o bit adhesivo
13. Sistemas de archivos distribuidos (*DFS*)
 - 13.1. *SAMBA*
 - 13.1.1. Evolución histórica de *SAMBA*
 - 13.1.2. Servicios proporcionados por *SAMBA*
 - 13.2. Sistema de archivos en red de *Sun Microsystems: NFS*
 - 13.2.1. Beneficios proporcionados por *NFS*
 - 13.3. Sistema de información de redes de *Sun Microsystems: NIS*
 - 13.4. Integrando *NIS* y *NFS*

Bibliografía

Básica

- Jack Tackett Jr. y David Gunter, “Linux Tercera Edición, Edición Especial”, Editorial Prentice Hall, 1998.
- Sebastián Sánchez Prieto, Óscar García Población, “UNIX y LINUX Guía Práctica Tercera edición”. Editorial Ra-Ma, 2005.

- Miguel Blanco Alonso, Jorge Fueyo Díaz, Benjamín López López, María Aida Martín Lucero, Antonio Paniagua Navarro, Ana M^a Pizarro Galán, Máximo Prudencio Conejo, Miguel Rodríguez Martín, Valentín Roldán Cuerpo, Francisco Torres Escobar, “Capítulo 16, el sistema de archivos, Descubre gnuLinEx”.

Complementaria

- Dee-Ann LeBlanc, “Administración de sistemas LINUX La biblia”. Editorial ANAYA MULTIMEDIA, 2001.
- Manuel Romero Velázquez, “El sistema de ficheros Ext3”.
<http://papeles.manoloromero.org/articulos/ext3.pdf>
- /dev/null.
<http://es.wikipedia.org/wiki//dev/null>
- /dev/zero.
<http://es.wikipedia.org/wiki//dev/zero>
- /dev/random.
<http://es.wikipedia.org/wiki//dev/random>
- Atributos de un archivo.
<http://www.ibiblio.org/pub/linux/docs/LuCaS/Manuales-LuCAS/doc-unixsec/unixsec-html/node57.html>
- Dispositivos, Sistema de archivos Linux, Herencia estándar.
http://wiki.xtech.com.ar/index.php/Dispositivos,_Sistema_de_archivos_Linux,_Herencia_est%C3%A1ndar#Atributos_de_archivos
- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.
- George Coulouris, Jean Dollimore, Tim Kindberg, “Sistemas Distribuidos Conceptos y Diseño Tercera edición”, Editorial Pearson Addison Wesley, 2005.
- “Manual de FreeBSD 6.1”, 1999.
<http://www.hacienda.go.cr/centro/datos/Articulo/Manual%20de%20FreeBSD.pdf>
- M. Carling, Stephen Degler, James Dennis, “Administración de Sistemas Linux, Guía Avanzada”, Editorial Prentice Hall, 2000.
- Alejandro Mauricio Valdés Jiménez, “Integrando NIS y NFS”.
<http://deb.utralca.cl/public/imagenes/nisnfs.pdf>

El sistema de archivos es la parte del sistema operativo responsable de la administración de los datos en un dispositivo de almacenamiento secundario. Todos los programas de Linux, las bibliotecas, los archivos del sistema y los archivos de usuario, residen en los sistemas de archivos. Por eso es fundamental llevar a cabo una correcta administración de los mismos y por ende mantenerlos sanos y organizados para evitar acabar pasando más tiempo buscando archivos y programas que administrando otros elementos del sistema.

1. Concepto de archivo y de sistema de archivos

Podemos definir de forma genérica el término archivo, como un conjunto de datos con un nombre asociado. La razón de asignar un nombre a cada archivo es que de este modo, tanto los usuarios como los programas, pueden hacer referencia a los mismos de una forma lógica. Los procesos o programas en ejecución disponen de un conjunto de funciones proporcionadas por el sistema operativo para poder manipular esos archivos. Ese conjunto de funciones se conoce con el nombre de llamadas al sistema o *system calls*. El concepto de llamada al sistema es más amplio, pues engloba también funciones relacionadas con la manipulación de procesos y dispositivos. Un proceso o programa en ejecución puede escribir datos en un archivo mediante la llamada al sistema *write* y leerlos más tarde, o bien dejarlos allí para que otros procesos puedan leerlos mediante la llamada al sistema *read*. También los procesos tienen la posibilidad de crear archivos, añadir o eliminar información en ellos, desplazarse dentro para consultar la información deseada, etc. a partir del correspondiente conjunto de llamadas al sistema.

El concepto de llamada al sistema ha sido comentado como un apunte informativo; el usuario final no tiene por qué ser consciente de la existencia de tales llamadas, ya que existen aplicaciones de más alto nivel que son las que las manipulan adecuadamente.

En cierto modo, se puede entender un archivo como una extensión del conjunto de datos asociados a un proceso, pero el hecho de que estos datos continúen existiendo aunque el proceso haya terminado, los hace especialmente útiles para el almacenamiento de información a largo plazo.

Un sistema de archivos debemos entenderlo como aquella parte del sistema responsable de la administración de los datos. El sistema de archivos debe proporcionar los medios necesarios para un almacenamiento seguro y privado de la información y, a la vez, la posibilidad de compartir esa información en caso de que el usuario lo desee.

Entre las características más relevantes del sistema de archivos de Linux podemos citar las siguientes:

- Los usuarios tienen la posibilidad de crear, modificar y borrar archivos y directorios.
- Cada archivo tiene definido tres tipos de acceso diferentes: acceso de lectura [*r*], acceso de escritura [*w*] y acceso de ejecución [*x*].
- A su vez, esos tres tipos de acceso pueden extenderse a la persona propietaria del archivo, al grupo al cual está adscrita dicha persona y al resto de los usuarios del sistema. Eso permite que los archivos puedan ser compartidos de forma controlada.
- Cada usuario puede estructurar sus archivos como desee, el núcleo no impone ninguna restricción.
- Proporciona la posibilidad de realizar copias de seguridad de todos y cada uno de los archivos para prevenir la pérdida de forma accidental o maliciosa de la información.

- Proporciona la posibilidad de cifrado y descifrado de información. Eso se puede hacer para que los datos sólo sean útiles (legibles) para las personas que conozcan la clave de descifrado.
- El usuario tiene una visión lógica de los datos, es el sistema el encargado de manipular correctamente los dispositivos y darle el soporte físico deseado a la información. El usuario no tiene que preocuparse por los dispositivos físicos, es el sistema el que se encarga de la forma en que se almacenan los datos en los dispositivos y de los medios físicos de transferencia de datos desde y hacia los mismos.

Cabe mencionar que en Linux uno de los principios básicos es la consideración de que todo flujo de *bits* constituye un archivo, cualquiera que sea su contenido. De esta manera, tanto una imagen como un texto son considerados como archivos; también tienen la misma consideración una carpeta, un disquete, una tarjeta de vídeo e, incluso, la conexión a una página web. Como todo es considerado un archivo, el software es el encargado de distinguir los diferentes tipos de archivos.

2. Los *inodos*

Los sistemas Linux son capaces de determinar qué bloques de datos contienen qué archivos o segmentos de archivo. Además, si los datos continúan en otro bloque, los sistemas Linux son capaces de resolver problemas como: saber cuántos bloques ocupa en total un archivo y saber el orden en que deben ser leídos dichos bloques de datos. Toda esta información está contenida en un objeto del sistema de archivos de Linux que se llama *inodo* (*Information Node*), que almacena los siguientes detalles de un archivo u objeto:

- El tipo de objeto definido por el *inodo*. Los tipos de objetos pueden ser: dispositivos (bloque o carácter), directorios, archivos (regular u oculto), tuberías, sockets o vínculos simbólicos (enlaces simbólicos o fuertes (duros)) (ver tabla 6.1).
- Los permisos del objeto.
- El propietario y el grupo para el objeto.
- Cómo es de grande el objeto, listado en bytes.
- Cuándo se creó el objeto.
- Cuándo se modificó el objeto por última vez.
- Indicios de dónde está físicamente localizado el objeto en el dispositivo. Si este objeto es un archivo que ocupa más de un bloque de datos, entonces se hace una lista de la ubicación de todos los bloques y el orden en el que se debería acceder a ellos.

Un *inodo* se nombra con un número que es único dentro de la partición o dispositivo donde reside. No hay necesidad de que ciertos *inodos* deban ir en grupos de bloque concretos. Pero los grupos de bloque siempre contienen los mismos bloques, incluso si la información en su interior cambia. La figura 6.1 muestra un ejemplo de cómo un *inodo* podría relacionarse con un sistema de archivos.

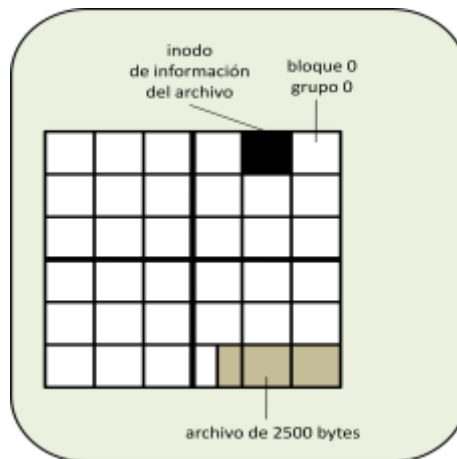


Figura 6.1: Utilizar *inodos* para asignar la información de los datos para los datos dentro de los grupos de bloque

Para identificar un archivo, Linux no utiliza su nombre sino su *inodo*, lo que supone que un mismo archivo se pueda guardar con nombres diferentes aunque su *inodo* sea el mismo y por ende también su contenido sea el mismo.

Por ejemplo:

```
$> touch prueba.txt
$> ls -li prueba.txt
1099187 -rw-r--r-- 1 gateway gateway 0 2008-04-20 03:06 prueba.txt
$> mv prueba.txt pruebaNUEVONOMBRE.txt
$> ls -li pruebaNUEVONOMBRE.txt
1099187 -rw-r--r-- 1 gateway gateway 0 2008-04-20 03:06 pruebaNUEVONOMBRE.txt
```

Para fines de organización, sólo es importante la información del grupo de bloque 0. Este factor existe porque cada grupo de bloques tiene una copia de la información de todos los demás grupos de bloque. Mientras el grupo de bloque 0 no esté dañado, todo va bien. Si lo está, los programas de reparación del sistema de archivos pueden recoger datos de otros grupos de bloque.

3. El superbloque

Un bloque especial contiene información sobre el conjunto del sistema de archivos, más que sobre sus componentes. Este segmento se llama *superbloque* y se encuentra disponible para echarle un vistazo con el comando *dumpe2fs* (por ejemplo: *sudo dumpe2fs -h /dev/sda4*). Algunos de los datos que contiene el *superbloque* incluyen:

- Cuántos *inodos* y bloques hay en total en el sistema de archivos.
- Cuántos bloques no se utilizan.
- Cuántos *inodos* no se utilizan.
- Cómo es de grande un bloque de datos individual. Este valor normalmente es de 1024 bytes.
- Cuántos bloques forman un grupo de bloques.

- Cuántas veces ha sido montado el sistema de archivos.
- Cuántas veces se puede montar el sistema de archivos antes de forzar una comprobación del mismo.
- En que grupos de bloques se almacena esta copia del *superbloque*.

Todo grupo de bloques contiene una copia del *superbloque* por propósitos de exceso. Al igual que con los datos del grupo de bloques, la copia del *superbloque* en el grupo de bloque 0 es la única que se utiliza en un sistema de archivos sano.

4. El sistema de archivos *ext2*

Como con otros sistemas operativos, el núcleo de Linux además de los archivos asociados con él, está almacenado en la unidad de disco duro, que es una unidad física. Dentro de cada controlador de disco duro hay particiones, que funcionan como una especie de controladores de disco duros virtuales. Después, cada partición, constituirá un sistema de archivos.

Linux es capaz de utilizar el sistema de archivos *ext2* (*second extended*), que fue creado especialmente para utilizarse con este sistema operativo (aunque actualmente se usa con mayor frecuencia el *ext3* al cual nos referiremos en la siguiente sección). Este sistema de archivos no está relegado a las unidades de disco duro, sino que está situado en cualquier medio que se utilice para almacenar datos de Linux. Entender cómo se divide el sistema de archivos es de gran ayuda cuando estamos intentando descifrar los comandos de manipulación del sistema de archivos.

Dentro del sistema de archivos *ext2*, los datos se almacenan en una serie de bloques de datos de idéntico tamaño (ver figura 6.2). Estos bloques son, generalmente, de 1024 bytes, aunque su tamaño puede ser cambiado mientras se confecciona el sistema de archivos. Si el archivo es de 10 bytes o de 1020 bytes, ocupa un bloque de datos que no puede ser utilizado para almacenar nada más. Aquellos archivos que sean de más de 1024 bytes, sean de 1025 o 2026 bytes, ocupan dos bloques de datos. Debido a la naturaleza de los sistemas de archivos, puesto que los archivos se agregan y se eliminan constantemente, estos bloques pueden estar, o no, físicamente uno al lado del otro.

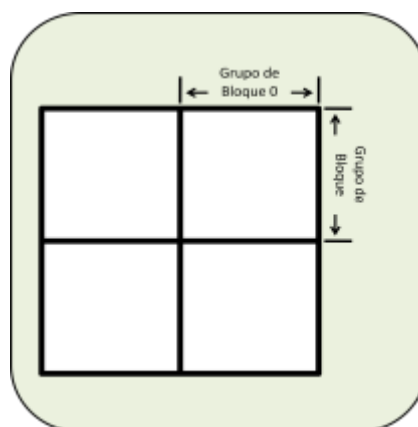


Figura 6.2: Dispositivo con un sistema de archivos *ext2*

Los bloques de datos individuales se organizan en grupos de bloque dentro del sistema de archivos. La integridad del sistema de archivos es una razón fundamental para utilizar grupos de bloque en lugar de un gran conjunto de bloques. Si todos los bloques de datos existen en un conjunto, el daño de una zona crítica lo daña todo. Los grupos de bloque no sólo descomponen los bloques de datos en sectores, sino que también dan flexibilidad y exceso de información.

5. El sistema de archivos *ext3*

El sistema de archivos *ext3* es muy similar al *ext2*, por lo tanto hereda de este último todas las características mencionadas en la sección anterior; además posee una característica de suma importancia para mantener sano y coherente nuestro sistema de archivos, esta característica es conocida como *journaling* (*JFS, Journaling File System*).

5.1. ¿Cómo surge *ext3*?

El continuo avance en el mundo de Linux y sus usos, ha ido creando la necesidad de nuevas herramientas. Un ejemplo de ello es el caso de la gestión de grandes cantidades de información, del orden de centenares de gigabytes, en los modernos sistemas de información. Cuando se manejan tales volúmenes de datos, es necesario que su manejo sea rápido y que las operaciones de mantenimiento no sean muy prolongadas. En el caso de utilizar sistemas tradicionales de archivos, el tiempo que se emplearía en comprobar un conjunto de particiones de estos tamaños sería de varias horas o incluso días. Pero si empleamos un sistema de archivos con *journaling* el tiempo de operación en escritura y lectura es bastante más reducido, pero sobre todo, lo que más se reduce es el tiempo de restauración del sistema tras una desconexión en la que el sistema de archivos no fue desmontado correctamente, en el caso de hacer uso de un sistema de archivos con *JFS* esto se reduce a unos cuantos segundos nada más.

Bajo esta línea ha sido desarrollado el sistema de archivos que actualmente se utiliza con mayor frecuencia en los sistemas Linux, el *ext3*, el cual es una nueva versión del sistema de archivos de uso común de GNU/Linux, el *ext2*. La principal novedad incorporada en *ext3* es el uso de *journaling*.

5.2. ¿Qué es *journaling*?

Se puede decir que *journaling* es un modo de trabajar o una funcionalidad que poseen ciertos sistemas de archivos, conocidos en español como *sistemas de archivos transaccionales*.

La idea de *journaling* implica dos conceptos a la hora de manejar la información, los *datos* y los *metadatos*. Por *datos* entendemos aquellos que representan la información que queremos almacenar, es decir, el contenido. Por otra parte los *metadatos* son aquellos que no nos sirven directamente, sino que se utilizan para que el sistema de archivos maneje los *datos*, por ejemplo: el tamaño de un archivo, su posición, sus atributos, etc.

Cada vez que un programa modifica los *datos* de un archivo, el sistema de archivos debe realizar modificaciones; por un lado en los *datos*, los cambios que se han realizado, y por otro en los *metadatos*, debido a que el tamaño del archivo, la fecha y puede que la ubicación y los permisos cambien. Para realizar estas operaciones un sistema de archivos sin *journaling* empezaría a modificar los *datos* y *metadatos* directamente hasta completar las modificaciones necesarias. Supongamos ahora que el sistema falla durante la realización de los cambios de *datos*, esto produce un archivo corrupto y lo más probable es que resulte inaccesible teniendo en cuenta que su estructura interna no coincidirá con una estructura válida. Sin embargo, si esto ocurre en el momento de modificación de los *metadatos*, en lugar de un solo archivo defectuoso, tendremos un sistema de archivos corrupto. Puede ocurrir que incluso no haya una estructura válida de los directorios y archivos, esto podría representar la pérdida de un directorio o de la totalidad del sistema de archivos.

Un sistema de archivos con *journaling* utiliza un procedimiento distinto al antes mencionado. Las operaciones de modificación de la información con *journaling* se realizan en una secuencia de tres pasos:

1. Registrar en un área especial llamada "*área de log*" los cambios que se van a hacer.
2. Realizar dichos cambios.
3. Eliminar del "*área de log*" los cambios una vez hechos.

Los *JFSs* funcionan usando dos áreas de espacio físico. Una para los datos, donde se almacenan tanto los *datos* como los *metadatos*, y otra para los *logs*, donde se escriben, al inicio de las operaciones, los cambios que va a realizar. Una vez hechos estos cambios tanto de *datos* como de *metadatos*, se confirman que se han hecho, esto último se hace usualmente borrando la información anterior del área de *log*. Esto permite que si se produce un fallo durante la escritura de los *metadatos*, tras el arranque del sistema, el *JFS* detecte en el área de *log* que no se completaron las modificaciones y use esa información para completarlas, esto permite que no se corrompa el sistema de archivos.

La implementación de *JFSs* conlleva un cierto precio asociado que en este caso es la velocidad. Los *JFSs* tienen una lentitud relativa a un sistema de archivos similar pero sin *JFSs*. Sin embargo, los *JFSs* destacan en rapidez en el manejo de grandes cantidades de archivos de pequeño tamaño, como es el caso de servidores web o servidores de bases de datos. Y en caso de que el sistema se caiga el tiempo para restablecer el sistema a su modo de operación normal se reduce considerablemente, lo que lo convierte en adecuado para implementaciones en sistemas donde se desea una alta disponibilidad.

6. El estándar de jerarquía del sistema de archivos

Parece haber una marcha sin fin de nuevas versiones de sistemas basados en Unix. No solamente las nuevas distribuciones de Linux aparecen de forma regular, sino que también hay FreeBSD y un número de variantes comerciales de Unix. Las empresas que quieren sacar software que trabaja con Unix y por ende con sistemas basados en Unix, han encontrado enormemente difícil desarrollar un código que fácilmente se instala en las versiones de Unix e incluso con variedades de Linux.

La primera respuesta a este problema resultó ser el Linux *File System STaNDard* (*FSSTND*), que se completó en 1994. Este estándar explica los directorios que deberían existir en un sistema de archivos de Linux y cómo se deberían utilizar. El propósito era hacer posible que los desarrolladores, en general, escribieran aplicaciones para Linux, sin tener que preocuparse por la distribución que debería seguir el software y, por lo tanto, consolidar el sistema operativo Linux. De alguna forma se alcanzó este objetivo, ya que existe un acuerdo general acerca del lugar en el que deberían ir muchos tipos de archivos. Sin embargo, por otro lado no, ya que por ejemplo, casi todas las distribuciones tienen diferentes modos de gestionar la información de arranque de redes. Puesto que las diferencias son mínimas y se pueden tratar leyendo las variables de entorno o buscando en localizaciones conocidas, el espíritu del objetivo todavía permanece intacto.

Una extensión natural del *FSSTND* era la de intentar hacer más fácil para los desarrolladores dirigir sus aplicaciones a más de una versión de Unix al mismo tiempo. Con este deseo creció el *File System Hierarchy Standard* (*FHS*). Este estándar pretende servir a más que a la comunidad Linux. Todo Unix está rodeado de él, hasta el lugar en el que deberían residir tipos particulares de datos y binarios en cualquier sistema de archivos Unix. Actualmente se utiliza *FHS* para definir los directorios principales y sus contenidos en los sistemas operativos Linux y en otros sistemas de la familia Unix.

El sistema de archivos de Linux se encuentra estructurado en forma de árbol invertido empezando a partir de un directorio inicial, denominado directorio raíz (ver figura 6.3) y representado por “/”, del que dependen los restantes directorios.

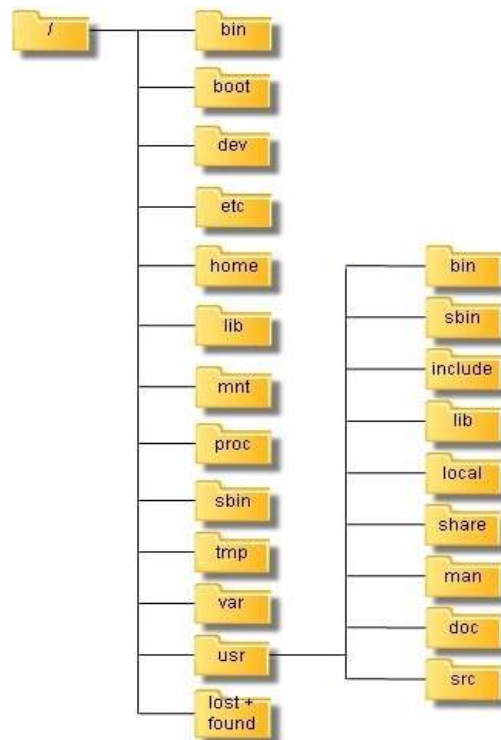


Figura 6.3: Esquema del árbol típico de directorios

En Linux los archivos se identifican en la estructura de directorios por lo que se conoce como *pathname* o camino, por ejemplo, la cadena */etc/passwd* identifica a *passwd* como un elemento que cuelga del directorio *etc* el cual a su vez cuelga del directorio raíz “/”. A partir de la cadena */etc/passwd* no podemos saber, a groso modo, si *passwd* es un archivo o es un directorio. Cuando el nombre del camino empieza por el carácter “/” se dice que el camino es absoluto, también disponemos de nombres de caminos relativos, por ejemplo, si nuestro directorio actual fuese */usr*, la cadena *bin/troff* identificaría al archivo o directorio */usr/bin/troff*. A esta cadena (*bin/troff*) se le conoce como camino relativo puesto que no comienza con el símbolo “/”. Por esta razón otra utilidad que posee “/” es separar directorios de subdirectorios y componer, de esta manera, la trayectoria o *path*, ya sea absoluto o relativo, de un archivo.

Cuando creamos un directorio, cuyos nombres son “.” (punto) y “..” (punto punto).

“.” es una entrada en el directorio que identifica al directorio mismo y “..” es una entrada al directorio padre, es decir, aquel directorio del cual cuelga el subdirectorio actual. Las cadenas “.” y “..” también pueden ser utilizadas en el nombre de un camino relativo. Por ejemplo, actualmente estamos colocados en */usr/lib*, la cadena *../include* identifica perfectamente al archivo o directorio */usr/include*.

Otros ejemplos:

Si consideramos el archivo *xterm*, éste puede ser referenciado tanto por su ruta absoluta como por su ruta relativa.

La ruta absoluta es algo que no depende de nuestra posición actual, y es de la forma:

/usr/bin/X11/xterm

La ruta relativa depende del directorio en que nos encontremos en cada instante. Por ejemplo, si estuviésemos colocados en el directorio `/usr/lib`, la ruta relativa de `xterm` sería:

```
../bin/X11/xterm
```

Los sistemas Linux hacen diferencia entre las letras mayúsculas y minúsculas tanto para nombres de archivos como para las rutas de los mismos. Así, el directorio `/usr/bin/X11` no es el mismo que `/usr/bin/x11`.

7. Algunos directorios interesantes

Todos los sistemas Linux, a diferencia de otros sistemas operativos, tienen una estructura de directorios estándar semejante a la representada en la figura 6.3. Seguidamente vamos a comentar algunos directorios que merecen una mención especial.

El directorio raíz “/”

Es un directorio pequeño, para facilitar su arranque desde otro *host* y dificultar que se pueda corromper. En él deben encontrarse sólo las herramientas que permitan arrancar, reparar y/o recuperar el sistema, además de los directorios básicos del sistema. Por tanto, en él debe encontrarse el *kernel* (`vmlinuz-x.x.x`) o un enlace simbólico al directorio `/boot`.

Directorio `/bin`

En este directorio se encuentran los programas ejecutables (*binarios*) esenciales para la administración del sistema; los comandos son ejecutables en la consola y estos pueden ser ejecutados por cualquier usuario.

Directorio `/sbin`

En este directorio se encuentran los ejecutables que son fundamentales para el funcionamiento del sistema, por lo que sólo el administrador del sistema debe tener acceso al mismo.

En este directorio se encuentran los comandos generales, los del encendido/apagado y reinicio del sistema, los de su mantenimiento y los referidos a redes.

Con la orden *which* podemos obtener la ruta completa de un determinado comando dentro del sistema, esta orden es muy útil ya que nos permite saber si un determinado comando está en `/bin` o `/sbin`.

Por ejemplo:

```
$> which route
/sbin/route
$> which netstat
/bin/netstat
```

Directorio `/boot`

En él radican archivos inalterables que se requieren para arrancar la máquina antes de que el núcleo asuma el control, tales como los archivos estáticos del cargador de arranque (*boot-*

loader). También se encuentran el *kernel* y un subdirectorio que contiene el gestor de arranque (*grub* o *lilo*, etc.) con su respectiva configuración.

Directorio */dev*

El nombre de este directorio proviene de *devices drivers* (controladores de dispositivos) porque es en este directorio donde se ubican los archivos que permiten la comunicación con los elementos del *hardware* instalados en nuestro ordenador. Aquí se encuentran las particiones del disco o discos (como */dev/hda* o */dev/sda*), las unidades de cd-rom scsi (como */dev/scd0*), también se hallan: las impresoras, los puertos serie, los puertos usb, el ratón, la tarjeta de sonido, etc.

Dentro del directorio */dev*, podemos encontrar archivos que poseen un comportamiento curioso e interesante de conocer como lo son:

/dev/null: Conocido como *null device* (periférico nulo), es un archivo especial que descarta toda la información que se escribe o redirecciona en él. A su vez, no proporciona ningún dato a cualquier proceso que intente leer de él, devolviendo simplemente un *EOF*. Generalmente se usa en *shell scripts* para redirigir el *stream* de salida de un proceso, o como, un archivo vacío que actúa como entrada para un *stream* de un proceso.

/dev/zero: Es un archivo especial que provee tantos caracteres *null* (ASCII NUL, 0x00; no el carácter ASCII "0", 0x30) como se lean desde él. Uno de los usos típicos es proveer un flujo de caracteres para sobrescribir información. Otro uso puede ser para generar un archivo "limpio" de un determinado tamaño. De la misma manera que */dev/null*, */dev/zero* actúa como fuente y sumidero de información. Todas las escrituras a */dev/zero* ocurren sin ningún efecto y todas las lecturas a */dev/zero* retornan tantos caracteres *NULs* como sean requeridos.

/dev/random: Es un archivo especial que sirve como un generador de números aleatorios, o un generador de números pseudo-aleatorios. Permite el acceso a ruido ambiental recogido de dispositivos y otras fuentes. La implementación utiliza hashes seguros en lugar de cifrados, fue diseñado bajo la premisa que cualquier hash o cifrado podría, eventualmente, ser débil por lo que el diseño es robusto frente a cualquiera de esas debilidades.

/dev/urandom: Es la contraparte de */dev/random*, este archivo reutiliza la fuente interna para producir más bits pseudo-aleatorios. La intención es servir como un generador de números pseudo-aleatorios criptográficamente seguro. Este puede ser utilizado en implementaciones que no necesiten de tanta seguridad.

Directorio */etc*

Contiene una serie de archivos de configuración y arranque del sistema. Aquí se incluyen subdirectorios como los de */etc/rc*, correspondientes a los guiones de inicialización del sistema, o archivos del tipo */etc/passwd*, donde están los usuarios del sistema.

También se ubican en */etc* subdirectorios utilizados para la configuración de determinados componentes del sistema, como, por ejemplo, */etc/X11*, que contiene los archivos de configuración del entorno gráfico *X Window*, con los gestores de ventanas y de arranque de los distintos escritorios.

Se encuentran en él archivos básicos, como *bashrc*, con la configuración de la *shell*, *inittab*, con la configuración de inicio o *fstab*, que contiene la relación de los puntos donde se montan las diferentes particiones y los sistemas de archivos que se utilizan en el sistema.

Directorios */home* y */root*

El directorio */home* contiene los subdirectorios personales de los usuarios. La existencia de diversos subdirectorios personales se debe a que Linux es un sistema multiusuario y, con esta configuración, es posible que varios usuarios puedan interactuar con el ordenador sin interferencias mutuas.

Por esta misma razón, el administrador tiene su propio directorio (*/root*) que, por seguridad, está separado del de los restantes usuarios.

Directorio */lib*

Contiene las bibliotecas compartidas de los binarios de los directorios */bin* y */sbin*, así como las compartidas por muchos programas, para reducir el espacio usado en el disco duro.

También se hallan aquí los módulos del *kernel*, que permiten el funcionamiento de muchos elementos del *hardware*.

Directorio */usr*

Es uno de los directorios más importantes y complejos del sistema, pues en él se encuentran los programas que utilizan los usuarios, con sus ejecutables, bibliotecas, referencias, iconos y documentación. Contiene archivos compartibles y otros que son de sólo lectura.

En */usr/bin*, se sitúan los ejecutables opcionales del sistema y de programas de uso común, como procesadores de texto o de tratamiento de imágenes. Por el contrario, sólo *root* puede acceder a los binarios del sistema no esenciales, contenidos en */usr/sbin*.

Cuando, en ocasiones, deseamos abrir un determinado archivo con un programa que no es el predeterminado por el sistema, la elección del ejecutable de la aplicación alternativa se efectúa en */usr/bin*.

Los ejecutables de los juegos y programas educativos se ubican en */usr/games*, en tanto que sus datos variables se hallan en */var/games*.

/usr/share contiene elementos y datos no modificables de los programas, así como información e iconos relativos a los mismos.

Otros manuales y documentaciones de los programas se encuentran en los subdirectorios específicos */usr/man* y */usr/doc*, mientras que sus bibliotecas se hallan en */usr/lib*. */usr/X11R6* es el subdirectorio donde se encuentran los programas que gestionan la interfaz gráfica para usuario.

Directorio */var*

Su nombre procede de *variable* porque contiene archivos y directorios de datos que cambian regularmente (son de carácter variable) como por ejemplo las colas de impresión o el correo no enviado (en */var/spool*).

Mientras que la información sobre el estado *variable* de las aplicaciones se sitúa en */var/lib*, en */var/log* se guardan los mensajes de registro generados por el sistema operativo y por diversos programas. Su utilidad radica en conocer los procesos y poder detectar problemas para prevenirlos o solucionarlos.

Algunos archivos de bloqueo se encuentran en `/var/lock`, mientras que algunos datos relevantes para determinados procesos que se ejecutan están en `/var/run`. En `/var/tmp` se ubican datos temporales que deben ser guardados entre reinicios del sistema y son utilizados para mantener a `/tmp` pequeño.

`/var/cache` contiene datos de aplicaciones en *cache*, como los paquetes que hemos instalado, y por lo tanto descargado de los repositorios, en el sistema con anterioridad, los cuales se hallan en `/var/cache/apt/archives`.

Directorio `/mnt`

Punto de montaje para aquellos sistemas de archivos que son montados temporalmente. Este directorio es usado mayormente por los usuarios. Sirve para montar discos duros y particiones de forma temporal en el sistema.

Directorio `/opt`

Contiene Paquetes de programas opcionales de aplicaciones estáticas, es decir, que pueden ser compartidas por los usuarios. Estas aplicaciones, utilizan el directorio de usuario para guardar sus configuraciones, y de esta forma, cada usuario puede tener una configuración diferente, de la misma aplicación.

Directorio `/lost+found`

Es un directorio que existe en todos los sistemas de archivos *ext2* y *ext3*, es donde el programa de chequeo del sistema coloca los bloques que ha encontrado y no ha podido ubicar, por ejemplo cuando existe una interrupción de suministro eléctrico. Por nombre tienen un número, porque los bloques colocados en este directorio se tratan de *inodos*.

Directorio `/proc`

Si listamos el contenido de este directorio con la orden `ls -F` obtendremos como resultado que las carpetas tienen, como fecha, la actual y, como hora, la del inicio de la sesión; además, sus contenidos son 0 bytes.

La razón estriba en que se trata de un sistema virtual de archivos mediante el cual el *kernel* se comunica con el usuario y le muestra los procesos que está ejecutando. El nombre corresponde al *pid* de un proceso. Estos archivos se hallan en la memoria y no físicamente en el directorio.

`/proc/kcore` es el único archivo, junto con los enlaces simbólicos presentes, cuyo tamaño es diferente de cero. Representa la memoria física que tiene el ordenador y, por tanto, su tamaño coincide con ésta, incrementada en algunos Megabytes.

Directorio `/tmp`

Espacio para todos aquellos programas que necesitan crear archivos transitorios.

8. Nombres de archivos y directorios

Aunque hemos tratado con distintos nombres de archivos y directorios, todavía no sabemos qué reglas se utilizan para nombrarlos.

Los nombres de los archivos pueden contener hasta 255 caracteres, aunque algunas versiones antiguas de Linux sólo permitían hasta 14. Los caracteres empleados pueden ser cualesquiera.

En la práctica, sin embargo, se suelen evitar aquellos caracteres del código ASCII que tienen un significado especial para el intérprete de órdenes (*shell*).

Como caracteres especiales para la *shell* podemos citar los siguientes:

? * > < / [] \ \$ “ () *etc.*

En realidad, se puede utilizar cualquiera de esos caracteres poniendo entre comillas simples (‘) el nombre del archivo; por ejemplo, ‘hola>mundo<.c’, pero resultará difícil acceder al archivo en la mayoría de los programas, y el archivo podrá presentar problemas para ser transferido a otros sistemas Linux.

Si queremos evitar problemas de interpretación por parte de la *shell*, no deberemos utilizar nombres de archivos como los que se indican seguidamente:

\$dinero\$
?datos
<desastre>
50/60_nombres

Aunque no son lo mismo los nombres de archivos que los nombres de rutas de acceso, después de todo, los directorios son considerados también como archivos. Es por esto que cuando coloquemos nombres a los directorios debemos recordar que estos también tienen las mismas limitaciones que los archivos normales.

8.1 Convenios en los nombres de los archivos

A pesar de que el nombre de un archivo puede elegirse, ciertas aplicaciones toman como convenio que los archivos con los cuales trabajan se diferencien del resto en algún rasgo identificador. Entre estas aplicaciones podemos citar los programas fuente escritos en un lenguaje de alto nivel. De este modo, un archivo que termine en *.c*, indica que contiene código fuente en lenguaje C. Si termina en *.f*, indica que contiene código fuente en FORTRAN; si acaba en *.p*, se trata de un programa escrito en Pascal, etc. Esto no impide que alguien llame a un juego, por ejemplo, *juego.p*, aunque no se corresponda con un programa fuente escrito en Pascal.

Los convenios anteriores no afectan a los programas que contienen código ejecutable. Tales programas pueden tener cualquier nombre, lo que despista mucho a las personas que están acostumbradas a trabajar con sistemas operativos en los que los archivos ejecutables tienen algún rasgo diferenciador del resto de los archivos.

Al hablar del nombre de los archivos no hemos mencionado el concepto de extensión, empleado en otros sistemas. En Linux un archivo puede no tener extensión, tener una, dos o *n*. Así pues, los siguientes nombres de archivos son perfectamente válidos en Linux:

programa.ejecutable.uno
prog.ver.1.1.0.3
holamundo.c.p.f

9. Tipos de archivos

Linux utiliza una amplia gama de archivos. De hecho, el sistema operativo, como lo mencionamos antes, trata todo como si se tratara de un archivo, desde los dispositivos como la

pantalla y la unidad de disco, hasta los “verdaderos archivo”, como los programas binarios o archivos de texto.

Existen cuatro tipos básicos de archivos: archivos normales, directorios, enlaces y archivos especiales. Hay varias clases de archivos normales, enlaces y archivos especiales, y un gran número de directorios estándar. Describiremos cada uno de ellos en la siguiente sección.

Podemos utilizar la orden *file* para determinar el tipo de archivo. *file* reconoce si el archivo es ejecutable, de texto, de datos y demás tipos.

Por ejemplo:

```
$> file holamundo.c
holamundo.c: ASCII C program text
$> file holamundo.o
holamundo.o: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.0,
dynamically linked (uses shared libs), not stripped
```

9.1. Archivos normales

Los archivos normales son con los que se trabaja la mayor parte del tiempo. Los archivos normales pueden contener texto, código fuente en lenguaje C, archivos de órdenes *shell*, programas binarios ejecutables y datos de naturaleza diversa. Para Linux, un archivo no es más que un archivo. La única diferencia es que Linux sabe cuáles de sus archivos están marcados como ejecutables. Los archivos ejecutables se pueden ejecutar directamente, siempre que contengan algo que ejecutar y que estén en la ruta de acceso de búsqueda. Básicamente, la ruta de acceso de búsqueda es una lista de nombres de rutas de acceso que se han especificado, en las que Linux busca para encontrar un archivo ejecutable.

Los archivos ejecutables son archivos binarios, es decir, archivos que ejecutan código máquina y archivos de secuencias *shell*.

9.2. Archivos de directorio

Los directorios son archivos que contienen los nombres de archivos y subdirectorios, así como punteros hacia esos archivos y subdirectorios. Los archivos de directorio son el único sitio donde Linux almacena nombres de archivos. Cuando se lista el contenido de un directorio con la orden *ls -l*, lo único que se hace en realidad es listar el contenido del archivo de directorio.

9.3. Directorios y discos físicos

Como vimos anteriormente en la sección **Los *inodos***, a cada archivo en Linux se le asigna un número único llamado *inodo*. El *inodo* se almacena en una tabla, llamada tabla de *inodos*, que se asigna cuando el disco está formateado. Cada disco físico o partición tiene su propia tabla de *inodos*. Un *inodo* contiene toda la información referente a un archivo, incluyendo la dirección de los datos en el disco y el tipo de archivo.

El sistema de archivos de Linux asigna el número de *inodo* 1 al directorio raíz. Con ello Linux conoce la dirección en disco del archivo del directorio raíz, que contiene una lista de nombres de archivos y directorios y sus números *inodo* correspondientes. Linux puede encontrar cualquier archivo en el sistema consultando una cadena de directorios, comenzando por el directorio raíz.

Linux navega por su sistema de archivos por medio del encadenamiento hacia arriba y hacia abajo del sistema de archivos de directorio. Cuando movemos un archivo a un directorio en otro disco físico, Linux detecta esto al leer la tabla de *inodos*. En este caso, el archivo se mueve al nuevo disco, donde se le asigna un nuevo *inodo*, antes de suprimirlo de donde estaba originalmente. Cuando suprimimos un archivo en realidad no se toca el archivo, sino que Linux marca ese *inodo* como libre y lo devuelve al conjunto de *inodos* disponibles. Luego se borra la entrada del archivo en el directorio.

9.4. Enlaces

Un enlace es un puente a un archivo o directorio perteneciente al sistema; una referencia que podemos poner en cualquier sitio que nos interese y que actúa como un acceso directo a cualquier otro. Este mecanismo nos permite acceder a carpetas o archivos de forma más rápida y cómoda, sin tener que desplazarnos por la jerarquía de directorios.

Por ejemplo:

Imaginemos que somos un usuario (*user1*) que necesita acceder frecuentemente al directorio */usr/share/man/man3*. En lugar de escribir el largo comando que nos situaría en el directorio en cuestión cada vez que necesitáramos desplazarnos a él, podemos crear un enlace en nuestro propio directorio que nos redireccione directamente hacia allí. El comando `ln -s /usr/share/man/man3 mmm` nos crearía este puente, que hemos llamado *mmm*. El usuario sólo debería escribir (desde su directorio */home*) `cd mmm` y automáticamente el sistema lo redirigirá hacia */usr/share/man/man3*. Es importante tener en cuenta que al hacer un `cd ..` para ir al directorio superior, volveríamos al directorio */home* y no a */usr/share/man*, ya que hemos accedido a él a partir de nuestro enlace. Podemos ver este esquema de forma gráfica en la figura 6.4.

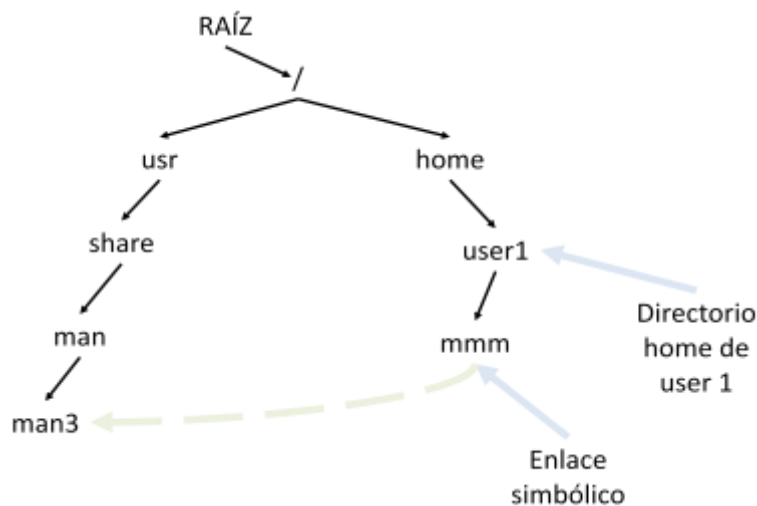


Figura 6.4: Esquema de un enlace simbólico

Al crear el enlace del ejemplo anterior hemos pasado el parámetro `-s` al comando `ln`. Ello indica que queremos crear un enlace *simbólico*. Los enlaces *simbólicos* significan que sólo estamos creando un apuntador o puente hacia el archivo o directorio, de forma que si borrásemos el archivo destino, el enlace no apuntaría a ninguna parte. Si no ponemos el parámetro `-s` se crearía lo que llamamos un enlace *fuerte* o *duro* (*hard link*) que, a diferencia del anterior, hace un duplicado del archivo. De hecho, internamente no es exactamente un duplicado, son como dos entradas que apuntan a los mismos datos. De este modo, si modificamos uno u otro, los dos quedan iguales. La ventaja de este tipo de enlace es que si borramos cualquiera de las dos copias del archivo la otra todavía se conserva. Este tipo de enlace no se utiliza demasiado porque complica la gestión y manipulación de los archivos, ya

que siempre es mejor tener una sola copia. Además, si hacemos un enlace *fuerte* de un directorio, todos los archivos y subdirectorios que contuviera también se deberían referenciar. Por esta razón sólo el *root* del sistema puede hacer enlaces fuertes de directorios. Otra diferencia es que con un enlace *simbólico* podemos ver a qué archivo estamos apuntando, mientras que con uno *fuerte* no podemos (debido al mecanismo que se utiliza internamente para ellos).

Un enlace *fuerte* sólo se puede crear entre archivos o directorios de una misma unidad debido al mecanismo interno que se utiliza para gestionarlos. En cambio un enlace *simbólico* puede señalar a otro archivo o directorio en el mismo disco, en otro disco o a un archivo o directorio en otra computadora.

9.5. Archivos especiales

En el sistema de archivos se representan todos los dispositivos físicos asociados a Linux, incluyendo discos, terminales e impresoras. La mayoría de los dispositivos, como vimos anteriormente, están ubicados en el directorio */dev*. Por ejemplo, si estamos trabajando con la consola del sistema, el nombre asociado de dispositivo es: */dev/console*. Si se estamos trabajando en una terminal estándar, el nombre del dispositivo puede ser: */dev/tty1*. Las terminales, o líneas seriales, se llaman dispositivos *tty* (que es la abreviatura de teletipo (*teletype*), el cual era la terminal original de Unix). Para saber cuál es el nombre de nuestro dispositivo *tty* podemos hacer uso de la orden *tty*. El sistema nos responderá con el nombre del dispositivo al que está conectado.

Las terminales e impresoras se denominan *dispositivos especiales por caracteres*. Pueden aceptar y producir una cadena de caracteres. Por otro lado, los discos almacenan datos en bloques que están direccionados por cilindro y sector. En un disco no se puede acceder sólo a un carácter, sino que hay que leer y escribir bloques completos. Esto mismo sucede normalmente en las cintas magnéticas. A este tipo de dispositivo se les denomina *dispositivos especiales por bloques*. Para complicar más las cosas, los discos u otros dispositivos especiales por bloques tienen que ser capaces de actuar como dispositivos orientados a caracteres, de modo que cada dispositivo de bloques tiene su correspondiente dispositivo especial por caracteres. Linux hace la traducción al leer los datos que se envían a un dispositivo por caracteres y traducirlos para un dispositivo por bloques. Esto se hace automáticamente sin la intervención del usuario.

Existe al menos otro tipo de dispositivo especial: un *FIFO* (memoria intermedia *first in first out*, primero en entrar primero en salir) que también se conoce como *conducción con nombre*. Los *FIFOs* parecen archivos normales; si se escribe en ellos aumentan de tamaño, pero cuando se leen disminuyen. Se utilizan principalmente en procesos del sistema para que muchos programas puedan enviar información sólo a un proceso del control.

En la siguiente tabla se resumen los tipos de archivos que existen en Linux según la naturaleza de su contenido.

Identificador	Tipo de archivo	Descripción
-	Archivo regular	Contenedor de información.
.	Archivo oculto	Archivo oculto que generalmente suelen contener archivos del sistema y de aplicaciones.
d	Carpeta o subcarpeta	Contiene una lista de nombres de archivo y sus correspondiente <i>inodos</i> .
b	Modo bloque	Dispositivos para el almacenamiento de datos (discos duros, particiones o CD-ROM).
c	Modo carácter	Dispositivos periféricos de entrada y salida (ratón, puertos serie, paralelos o USB).

Identificador	Tipo de archivo	Descripción
l	Enlace	Puente a un archivo o directorio perteneciente al sistema (enlace simbólico).
p	Tubería	Conexiones entre programas.
s	Socket	Conexiones de red.

Tabla 6.1: Tipos de archivos de Linux según la naturaleza de su contenido

10. Atributos existentes en el sistema de archivos de Linux

En los sistemas de archivos *ext2* y *ext3* de Linux existen ciertos atributos que nos son de utilidad a la hora de incrementar la seguridad de nuestro sistema. Estos atributos son los mostrados en la siguiente tabla.

Atributo	Descripción
A	Cuando se accede al archivo, la fecha de acceso no se modifica (<i>Don't update Atime</i>).
S	Grabar inmediatamente las modificaciones de los archivos en el disco, atributo interesante en los archivos de seguridad (<i>Synchronous updates</i>).
a	Solamente se puede agregar contenido al archivo y este no podrá ser eliminado. Sólo <i>root</i> puede modificar este atributo (<i>append only</i>).
c	Archivo comprimido (<i>compressed file</i>).
i	El archivo queda inmutable. Un archivo inmutable es aquel que no se puede copiar, borrar ni renombrar (<i>immutable file</i>).
d	Al hacer uso del comando <i>dump</i> para backup se omitirá el archivo que posea este atributo (<i>No dump</i>).
s	Cuando se borre el archivo este se llenará con ceros (<i>secure deletion</i>).
u	Cuando se elimina un archivo se guarda el contenido del mismo (<i>undeletable file</i>).
D	Equivalente a S, pero se aplica a nivel de directorio.
E	Muestra que un archivo comprimido tiene un error de compresión.
I	Se aplica a directorios para indicar que estos están indexados con un árbol hash.
j	Los datos se escriben primero al <i>journal</i> del sistema <i>ext3</i> antes de escribirse en el propio archivo.
T	Sirve para considerar a un directorio como si estuviera en el primer nivel de directorios.
t	Hace que el archivo no posea un fragmento al final combinado con otro archivo.
X	El contenido crudo de un archivo comprimido se puede acceder directamente.
Z	El archivo comprimido está sucio.

Tabla 6.2: Atributos existentes en el sistema de archivos de Linux

De todos los atributos mostrados en la tabla 6.2, de cara a la seguridad del sistema, existen algunos que no interesan demasiado, pero hay otros que sí. Uno de los atributos interesante (quizás el que más interesante) es “a”; tan importante es que sólo el administrador tiene el privilegio suficiente para activarlo o desactivarlo. El atributo “a” sobre un archivo indica que este sólo se puede abrir en modo escritura para añadir datos, pero nunca para eliminarlos. ¿Qué tiene que ver esto con la seguridad? Muy sencillo: cuando un intruso ha conseguido el privilegio suficiente en un sistema atacado, lo primero que suele hacer es borrar sus huellas; para esto existen muchos programas (denominados *zappers*, *rootkits*) que, junto a otras funciones, eliminan estructuras de ciertos archivos de *log*. Así consiguen que cuando alguien ejecute algunos comandos como: *last*, *who*, *users*, *w* o similares, no vean el rastro de la conexión que el atacante ha realizado a la máquina; evidentemente, si estos archivos de *log* poseen el atributo “a” activado, el atacante y sus programas lo tienen un poco más difícil para borrar datos de ellos.

Otro atributo interesante es “i” (*immutable file*); un archivo con este atributo activado no se puede modificar de ninguna forma, ni añadiendo datos ni borrándolos, ni eliminando el archivo, ni tan siquiera enlazándolo mediante la orden *ln*. Igual que sucedía antes, sólo el administrador puede activar o desactivar el atributo “i” de un archivo. Podemos aprovechar esta característica en los archivos que no se modifican frecuentemente, por ejemplo muchos de los contenidos en */etc* (archivos de configuración, scripts de arranque, incluso el propio archivo de contraseñas si el añadir o eliminar usuarios tampoco es frecuente en nuestro sistema); de esta forma conseguimos que ningún usuario pueda modificarlos incluso aunque sus permisos lo permitan. Cuando activemos el atributo “i” en un archivo hemos de tener siempre en cuenta que el archivo no va a poder ser modificado por nadie, incluido el administrador (*root*), y tampoco por los programas que se ejecutan en la máquina; por tanto, si activamos este atributo a un archivo de *log*, no se grabaría ninguna información en él, lo que evidentemente no es nada conveniente. También tenemos que recordar que los archivos tampoco van a poder ser enlazados, lo que puede ser problemático en algunas variantes de Linux que utilizan enlaces duros para la configuración de los archivos de arranque del sistema. Por estos motivos debemos tener especial cuidado a la hora de elegir a que archivos vamos a establecerles el atributo “i”.

Otro atributo que también puede ayudar a implementar una correcta política de seguridad en el sistema, aunque menos importante que los anteriores, es “s”. Si borramos un archivo con el atributo “s” activo, el sistema va a rellenar sus bloques con ceros en lugar de efectuar un simple *unlink()*, para así dificultar la tarea de un atacante que intente recuperarlo; realmente, para un atacante experto esto no supone ningún problema, simplemente un retraso en sus propósitos: ya que esta comprobado que un simple relleno de bloques mediante *bzero()* no hace que la información sea irrecuperable.

La regla general como buenos administradores es que debemos tener en cuenta que establecer atributos a algunos archivos o directorios puede llegar a **incrementar la seguridad** del sistema.

Ya hemos tratado con los atributos existentes, ahora veremos cómo activarlos o desactivarlos, y también cómo podemos ver su estado. Para lo primero podemos hacer uso de la orden *chattr*, que recibe como parámetros el nombre del atributo junto a un signo “+” o “-“, en función de si deseamos activar o desactivar el atributo, y también el nombre del archivo correspondiente. Si lo que deseamos es visualizar el estado de los diferentes atributos, utilizaremos la orden *lsattr*, cuya salida indicará con la letra correspondiente cada atributo del archivo o un guión “-” en el caso de que el atributo no esté activado.

Por ejemplo, agregar inmutabilidad a un archivo:

```
$> touch prueba.txt
$> lsattr prueba.txt
----- prueba.txt
$>sudo chattr +i prueba.txt
$> lsattr prueba.txt
---i----- prueba.txt
$> rm prueba.txt
rm: ¿borrar el archivo regular vacío `prueba.txt' protegido contra escritura? (s/n) s
rm: no se puede borrar `prueba.txt': Operación no permitida
```

11. Propiedad y permisos de los archivos

La propiedad y los permisos son también un punto básico en la seguridad del sistema. Es importante establecer correctamente estas opciones, incluso aunque seamos los únicos que utilicemos el sistema, ya que pueden suceder cosas extrañas si no se hace. Para los archivos que pueden crear y utilizar diariamente los usuarios, estos conceptos se pueden establecer sin tener que pensar mucho (aunque sigue siendo útil conocer dichos conceptos). Para la administración del sistema, no es tan fácil. Asignar una propiedad o permiso erróneo puede conducir a hechos lamentables, como por ejemplo el que un usuario no pueda ser capaz de leer su propio correo electrónico.

En general, el mensaje: *Permission denied (o permiso denegado)*; Significa que alguien ha asignado una propiedad o permiso que limita su acceso más de lo deseado.

11.1. ¿Qué significan los permisos?

Los permisos se refieren a la forma en que cualquier usuario puede utilizar un archivo. En Linux existen tres tipos de permisos:

- **Lectura [r]:** Este permiso significa que podemos ver el contenido de un archivo.
- **Escritura [w]:** Este permiso significa que podemos modificar o eliminar un archivo.
- **Ejecución [x]:** Este permiso significa que podemos ejecutar el archivo como un programa.

Cuando se crea un archivo, el sistema asigna algunos permisos predeterminados que funcionan bien normalmente. Por ejemplo, nos proporciona el permiso de lectura y el de escritura, pero el resto de personas sólo tienen permiso de lectura. Si tenemos tendencia a ser obsesivos, podemos establecer todo para que el resto de usuarios no tengan ningún tipo de permiso de acceso a ninguno de nuestros archivos. Asimismo, la mayoría de utilidades saben cómo asignar permisos. Por ejemplo, cuando el compilador crea un programa ejecutable, automáticamente le asigna un permiso de ejecución.

No obstante, los valores predeterminados a veces no funcionan. Por ejemplo, si creamos una secuencia de comandos de *shell* o un programa en *Perl*, tendremos que asignarnos a nosotros mismos un permiso de ejecución para poder ejecutarlo.

Los permisos tienen distintos significados para un directorio:

- El permiso de **lectura** significa que podemos listar el contenido de dicho directorio.
- El permiso de **escritura** significa que podemos añadir o eliminar archivos en dicho directorio.
- El permiso de **ejecución** significa que podemos listar información sobre los archivos de dicho directorio.

No debemos preocuparnos por la diferencia existente entre el permiso de lectura y el de ejecución en los directorios ya que, básicamente, van unidos. Debemos asignar ambos o ninguno.

Debemos tener en cuenta que si permitimos a otras personas añadir archivos a un directorio, también estaremos permitiéndoles eliminar archivos. Los dos privilegios van unidos cuando se asigna un permiso de escritura. Sin embargo, existe una forma de permitir a otros usuarios

compartir un directorio y evitar que eliminen archivos de unos y otros (ver la sección *El Sticky bit o bit adhesivo*).

11.2. Propietarios y grupos

Ahora la gran pregunta es, ¿Quién tiene dichos permisos? Para permitir que los usuarios trabajen juntos, Linux tiene tres niveles de permisos: propietario, grupo y otros. El nivel “otros” incluye a todo aquel que tiene acceso al sistema y que no es ni propietario ni miembro del grupo. La idea que se esconde tras los grupos es proporcionar acceso a un conjunto de usuarios, como un equipo de programación, a determinados archivos. Por ejemplo, un programador que está creando un código fuente puede reservarse para sí mismo el permiso de escritura pero permitir a los demás miembros de su grupo tener un acceso de lectura a través de un permiso de grupo. En cuanto a “otros”, no tendrá ningún tipo de permiso para que personas externas al equipo no puedan ver los archivos.

Cada archivo tiene un propietario y un grupo. Normalmente el propietario es el usuario que ha creado el archivo. Asimismo, todos los usuarios pertenecen a un grupo predeterminado y dicho grupo se asigna a todos los archivos creados por el usuario. Mediante el cambio del grupo asignado un archivo, puede proporcionar acceso a cualquier grupo de personas que desee.

A continuación vamos a examinar algunos archivos típicos y comprobar cuáles son los permisos que tienen asignados.

La figura 6.5 muestra un programa ejecutable típico. La salida ha sido generada ejecutando la orden *ls* con la opción *-l*.

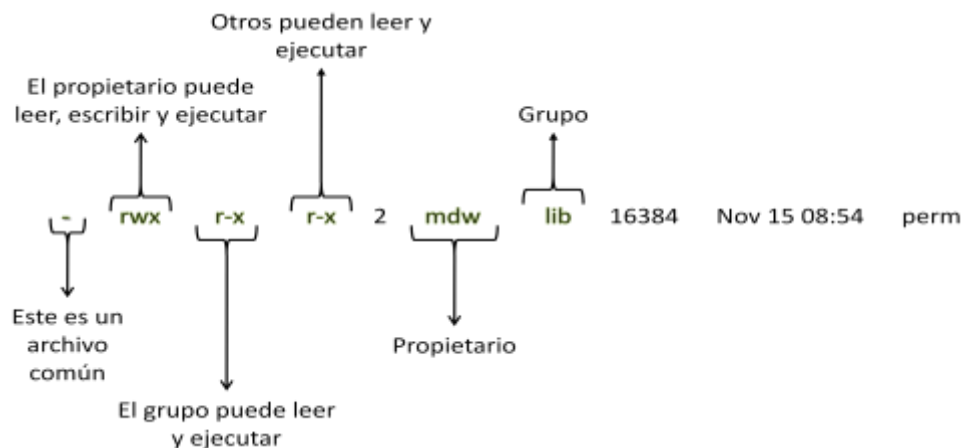


Figura 6.5: Mostrar la propiedad y los permisos

De la figura 6.5 podemos destacar dos hechos importantes: los propietarios del archivo son, *mdw*, y el grupo *lib* (quizá un grupo creado por los programadores que trabajan en las bibliotecas). Pero la información importante sobre los permisos se cifra en el conjunto de letras de la parte izquierda.

El primer carácter es un guión, que indica que se trata de un archivo común. Los tres siguientes bits se aplican al propietario; como cabría esperar, *mdw* tiene los tres permisos. Los siguientes tres bits se aplican a los miembros del grupo: estos pueden leer el archivo (algo poco útil para un archivo binario) y ejecutarlo, pero no pueden escribir en él porque el campo que debe contener una “w” contiene en su lugar un guión “-”. Los últimos tres bits se aplican a “otros”; y estos tienen los mismos permisos que el grupo.

Otro ejemplo:

```
$> ls -l simc.c
-rw-rw-r-- 1 kalle kalle 12577 Apr 30 13:13 simc.c
```

El listado muestra que el propietario tiene permisos de lectura y escritura “*rw*” así como el grupo. El resto de usuarios en el sistema sólo tienen privilegios de lectura. Ahora supongamos que compilamos el archivo *simc.c* para crear un programa ejecutable. El compilador *gcc* crea el archivo *simc*:

```
$> gcc -o simc simc.c
$> ls -l simc simc.c
total 36
-rwxrwxr-x 1 kalle kalle 19365 Apr 30 13:14 simc
-rw-rw-r-- 1 kalle kalle 12577 Apr 30 13:13 simc.c
```

Además de los bits de lectura y escritura, *gcc* ha establecido el bit ejecutable (*x*) para el propietario, el grupo y otros en el archivo ejecutable.

12. Puntos adicionales del sistema de archivos

En esta sección explicaremos algunos puntos acerca de los permisos que se pueden asociar a los archivos o directorios, los cuales son de gran utilidad para la seguridad de nuestro sistema.

12.1. Máscara frente a umáscara

La *máscara* de un objeto del sistema de archivos es la misma que su permiso. Los dos términos son equivalentes. Otro objeto útil es la *umáscara* (*user file-creation mode mask* o máscara del modo de creación de archivos de usuario). Este término se refiere a los bits que no están activados en la máscara o en los bloques de permiso.

La *umáscara* es una herramienta increíblemente útil para un administrador de sistemas. Muchos usuarios no comprenden los permisos, y por ende nunca los alteran cuando crean archivos y directorios dentro de su directorio */home*. Cuando los usuarios hacen esto con permisos débiles por defecto, este hecho puede causar problemas de seguridad. Por consiguiente, cambiar la *máscara* de creación por defecto configurando la *umáscara* por defecto a una configuración más fuerte, nos puede ahorrar muchos dolores de cabeza.

Para ejemplificar el proceso tomemos un archivo creado por un usuario (por ejemplo: *gateway*) con una *umáscara* de 133 que produce archivos con permisos de 644.

```
$> ls -l archivo
-rw-r--r-- 1 gateway gateway 12577 Apr 30 13:13 archivo
```

El modo resultante es que el dueño (*gateway*) tiene permisos de lectura y escritura; y el grupo y los demás sólo tienen permiso de lectura. Una forma de darse cuenta de la forma en que funciona el *umask* es tener en cuenta que el valor (2) inhabilita el permiso de escritura mientras que el valor (7) inhabilita los permisos de lectura, escritura y ejecución.

A continuación se muestra una tabla con los valores comúnmente usados para el *umask*.

Umask	Accesos del usuario	Accesos del grupo	Accesos de los otros
000	Todos	Todos	Todos
002	Todos	Todos	Lectura y ejecución
007	Todos	Todos	Ninguno
022	Todos	Lectura y ejecución	Lectura y ejecución
027	Todos	Lectura y ejecución	Ninguno
077	Todos	Ninguno	Ninguno

Tabla 6.3: Valores comúnmente usados para el *umask*

12.2. Establecer ID de usuario y de grupo (*SUID* y *SGID*)

Existen ocasiones en que los usuarios necesitan ejecutar algún comando que requiere de privilegios. Un ejemplo de esto es el uso del comando *passwd* para cambiar la contraseña. Sería un error darle a los usuarios los privilegios necesarios para que puedan ejecutar esta clase de ordenes ya que el usuario podría cambiarse de grupo o crear una cuenta con privilegios de *root*. Para que esto no suceda en Linux se implementó un sistema por el cual un comando que cuente con *SUID* o *SGID* puede ser ejecutado con los privilegios del dueño y/o grupo del programa.

Cada usuario esta identificado por el sistema con un número de identificación tanto para él, como para el grupo. Este número se denomina *UID* (*user ID*) para el caso de los usuarios y *GID* (*group ID*) para el caso de los grupos. Lo que se efectúa con el sistema *SUID* es una adquisición temporal de un *UID* o *GID* distinto al propio cuando se esta ejecutando el comando. Cuando un comando cambia de *UID* se denomina *SUID* (*set user ID*) y cuando cambia de *GID* se denomina *SGID* (*set group ID*) Un comando puede ser *SUID* y *SGUID* al mismo tiempo.

Para darnos cuenta si un comando es *SUID* basta con hacer un listado largo con el comando *ls -l* y se verá que donde tendría que estar una “x”, que asigna permisos de ejecución, aparece una letra “s”. Ejemplo:

```
$> ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 29104 2006-12-19 14:35 passwd
```

12.3. El Sticky bit o bit adhesivo

En los antiguos sistemas Unix la memoria era algo esencial y escasa debido a su alto costo. Para hacer un uso más productivo de la misma se empleo una tecnología que mantenía parte de programas esenciales en el área *swap* (intercambio) de memoria para que pudieran ser usados más rápidamente, dado que si estos no estuvieran en el área de *swap* entonces se tendrían que ir a buscar al disco y por ende se tardaría más. Estos programas se los marcaba con un bit especial, un *bit adhesivo* o *sticky bit*. Los programas así marcados eran los que valía la pena mantener en el área de *swap* ya que esas partes del programa que se guardaban en memoria también podían ser usadas por otros.

En los sistemas operativos Linux modernos el *sticky bit* encendido en un directorio significa que solo el dueño del archivo tiene derecho a borrarlo.

Directorios públicos con permisos de escritura “w” deben tener el *sticky bit* encendido ya que de otra manera todos los que accedan a dicho directorio pueden borrar archivos aunque no sean de ellos. Al encender el bit solamente los dueños podrán borrar sus archivos.

Por ejemplo:

```
$> ls -la /tmp/
drwxrwxrwt 12 root root 4096 2008-04-24 02:35 .
drwxr-xr-x 21 root root 4096 2008-04-17 10:03 ..
drwxrwxrwt 2 gateway gateway 4096 2008-04-24 00:38 .esd-1000
drwx----- 3 gateway gateway 4096 2008-04-24 00:38 gconfd-gateway
drwxrwxrwt 2 root root 4096 2008-04-24 00:38 .ICE-unix
drwxrwxrwt 2 root root 4096 2008-04-24 00:36 .X11-unix
$> rm -rf /tmp/.X11-unix
rm: no se puede borrar `/tmp/.X11-unix/X0': Operación no permitida
```

13. Sistemas de archivos distribuidos (DFS)

Un sistema de archivos distribuido almacena archivos en uno o más computadores denominados servidores de archivos y los hace accesibles a otros computadores denominados clientes estos manipulan dichos archivos como si estuvieran ubicados localmente en su máquina.

Existen muchas ventajas de hacer uso de servidores de archivos dentro de las cuales podemos encontrar:

- Los archivos están más accesibles debido a que desde varios computadores se puede acceder a los servidores de archivos.
- Compartir los archivos de una única localización es más sencillo que distribuir copias de los archivos a todos los clientes.
- Las copias de respaldo y la seguridad son más fáciles de manejar cuando sólo hay que tener en cuenta a los servidores de archivos.
- Los servidores de archivos pueden ofrecer un gran espacio de almacenamiento; algo que sería costoso y poco práctico de suministrar en cada uno de los clientes.

Sistemas de archivos distribuidos basados en Linux son utilizados para centralizar la administración de discos y proveer la facilidad de compartir archivos de forma transparente por la red. Paquetes de Linux que proveen *DFS* casi siempre incluyen el cliente y el servidor. Un servidor *DFS* comparte archivos locales en la red; un cliente *DFS* monta archivos compartidos localmente.

En las siguientes secciones vamos a analizar algunos de los métodos más importantes en el mundo de Linux, en cuanto a *DFS* se refiere. Primero vamos a analizar *SAMBA*, que utiliza protocolos de sistemas de red de Microsoft Windows para permitir a los usuarios de un sistema leer y escribir archivos en otro sistema y para enviar tareas a las impresoras en sistemas remotos. La ventaja de utilizar *SAMBA* es que se pueden integrar Linux y Unix sin problemas con los sistemas Microsoft, tanto clientes como servidores. Los protocolos de los sistemas de red de Microsoft Windows se pueden utilizar para compartir archivos entre sistemas de Linux.

También analizaremos dos protocolos desarrollados por *Sun Microsystems*, *NFS* y *NIS*, los cuales se utilizan en los sistemas Linux y Unix desde hace décadas. El sistema de archivos de red (*NFS*, *Network File System*) permite a los sistemas compartir archivos entre sistemas Linux y Unix de una forma muy similar a *SAMBA*. El sistema de información de red (*NIS*, *Network Information System*) permite que la información del usuario se guarde en un lugar y puedan acceder a ella múltiples sistemas y así no tener que actualizar todos los sistemas cuando se

cambia un usuario o una contraseña. Aunque *NIS* no es una herramienta para compartir archivos e impresoras, la analizaremos porque comparte algunos elementos con *NFS* y además puede facilitar la administración de *NFS*, ya que *NIS* permite a cada usuario tener el mismo número de cuenta en todos los sistemas.

NFS y *NIS* son útiles en sitios donde sólo están conectados sistemas Linux y variantes de Unix. Se han creado versiones para sistemas de Microsoft, pero no son particularmente robustos y nunca han sido muy populares. Microsoft proporciona una implantación de cliente y servidor *NFS* complementaria para sistemas Windows, que no se utilizan a menudo a pesar de ser gratuitos.

Además de los protocolos de red de MS Windows y *NFS*, existen otros protocolos para compartir archivos e impresoras muy conocidos. Linux admite compartir archivos e impresoras al estilo de *NetWare* utilizando protocolos *IPX*, compartir archivos e impresoras basados en Macintosh utilizando protocolo *Apple-Talk*, compartir archivos sobre protocolos como: compartir archivos sobre *SSH* (*FISH*, *File Sharing over SSH*); así como servicios de archivos basados en *WebDAV*.

13.1. SAMBA

La revolución del software de código libre todavía no ha concluido, por lo tanto, como resultado, actualmente siguen existiendo muchos escritorios y sistemas de servidores de Windows. Aunque muchos pensemos que el mundo pronto utilizará nada más que escritorios Linux, la realidad indica algo totalmente diferente: los escritorios Windows seguirán durante mucho tiempo entre nosotros. Por tanto, la capacidad de intercambiar archivos entre los sistemas Windows y Linux es muy importante. La capacidad para compartir impresoras es igualmente importante.

SAMBA es una familia de aplicaciones muy flexible y escalable que permite a un usuario de Linux leer y escribir archivos ubicados en estaciones de trabajo de Windows y viceversa. Puede que deseemos utilizar *SAMBA* sólo para que los archivos de Linux se encuentren disponibles para un solo cliente de Windows (como cuando ejecutamos Windows en un entorno de máquina virtual sobre un portátil con Linux). Pero también podemos utilizar *SAMBA* para implantar un servidor de archivos e impresoras de alto rendimiento para una red que tiene cientos de clientes de Windows.

13.1.1. Evolución histórica de *SAMBA*

A mediados de los años 80, *IBM* y *Systec* desarrollaron un sencillo sistema para proporcionar servicios de red denominados *NetBIOS* (*Network Basic Input Output System*). Dicho sistema estaba orientado a trabajar con pequeñas redes aisladas, sin capacidad de interconexión entre sí, es decir, no contemplaba la posibilidad de encaminamiento de datos a través de redes. *MS-DOS* incluyó la posibilidad de redireccionar el sistema de entrada y salida de los discos hacia la interfaz de *NetBIOS*, de forma que el contenido de los sistemas de archivos fuera accesible a través de la red. El protocolo para compartir archivos a través de la red se denominó *SMB* (*Server Message Block protocol*). Actualmente a este protocolo se le conoce como *CIFS* (*Common Internet File System*).

El segundo paso fue ampliar los servicios proporcionados por *NetBIOS* para que pudieran operar sobre redes Ethernet y Tokenring. El resultado fue *NetBEUI* (*NetBIOS Enhanced User Interface*). También se desarrolló software para emular *NetBIOS* sobre protocolos de mayor nivel, como *IPX* o *TCP/IP*. Este último es muy interesante porque permite enviar paquetes *NetBIOS* a través de redes interconectadas mediante *routers* o encaminadores. *NetBIOS* se desarrolló para trabajar en pequeñas redes aisladas, así es que la solución fue traducir los nombres de *NetBIOS* (dieciséis bytes para denominar un equipo) a direcciones *IP*. Más tarde

Microsoft añadió alguna funcionalidad adicional al paquete *SMB*: el servicio de anuncio (*browsing*) y un servicio de autenticación centralizada denominado *Dominio NT*, que se incluyó por primera vez en Windows NT 3.51 (Windows NT Domain Controller).

En esa misma época, Andrew Tridgell (ver figura 6.6) estaba trabajando en un software que permitiera acceder a un PC con sistema operativo *MS-DOS*, a un sistema de archivos residente en una máquina Unix. Esa parte no era un problema porque existía un paquete para utilizar *MS-DOS* con sistemas *NFS*. El problema era la coexistencia en *MS-DOS* de dos protocolos de red distintos: *NFS* y *NetBIOS*. Andrew Tridgell escribió un *sniffer* de paquetes de forma que pudiera hacer ingeniería inversa sobre el protocolo *SMB*, ya que este protocolo era y sigue siendo propietario de Microsoft. Cuando las primeras versiones estuvieron disponibles, una compañía de software reclamó los derechos sobre el nombre dado a su sistema servidor de archivos (*SMB*). Para solucionar este problema, Andrew Tridgell buscó una lista de palabras que contuvieran las letras *SMB*, ése es el origen del nombre actual: *SAMBA*.



Figura 6.6: Andrew Tridgell

13.1.2. Servicios proporcionados por *SAMBA*

El servicio *SAMBA* está formado por dos programas que se ejecutan como demonios en el sistema: *smbd* y *nmbd*. Su objetivo es proporcionar cuatro servicios claves del protocolo:

- Servicios sobre archivos e impresoras.
- Autenticación y autorización.
- Resolución de nombres.
- Anuncio de servicios en la red (*browsing*).

Los servicios sobre archivos e impresoras los proporciona *smbd*. Éste también se encarga de proporcionar servicios de autenticación y autorización a través de dos modos de trabajo: modo compartido (*share mode*) y modo de usuario (*user mode*). El primero permite compartir un recurso utilizando una única contraseña para todo aquel que quiera acceder. En el segundo cada usuario tiene su propia contraseña y el administrador puede autorizar o denegar el acceso a cada usuario independientemente.

El concepto de Dominio NT añade un mecanismo adicional de autenticación, que consiste en que un usuario se autentica una única vez y, una vez hecho esto, tiene acceso a todos los servicios para los que esté autorizado dentro de un dominio. Este servicio lo proporciona un controlador de dominio (*domain controller*). Así, un dominio es un conjunto de máquinas que comparten el mismo controlador de dominio.

Los otros dos servicios, resolución de nombres y anuncio de servicios, los proporciona *nmbd*. El objetivo es propagar y controlar una lista de nombres *NetBIOS* de equipos. La resolución de nombres se puede llevar a cabo de dos formas, mediante difusión (*broadcast*) y punto a punto. La primera es la solución más cercana a la implementación original. Cuando una máquina quiere conocer la dirección IP de un equipo, difunde su nombre a través de toda la red a la espera de que el aludido responda con su dirección IP. Esto puede generar algo de tráfico en la red, pero siempre confinado a la red local. El segundo mecanismo implica utilizar un servicio conocido como *NBNS* (*NetBIOS Name Server*). Microsoft llamó a su implementación de este servicio *WINS* (*Windows Internet Name Server*). Cuando una máquina arranca, registra su nombre y su dirección IP en este servidor, de forma que cuando quiere encontrar la dirección IP de una máquina a través de su nombre consulta en este mismo servicio. La ventaja de esta aproximación es que las máquinas situadas en redes distintas pueden compartir el mismo servidor *NBNS*, por lo tanto, el servicio no está limitado únicamente a las máquinas confinadas a la misma red local.

Por último, el anuncio (*browsing*) consiste en hacer saber a los demás participantes qué servicios comparte un determinado equipo. Inicialmente todos los equipos que componen una red llevan a cabo un proceso de selección para determinar quién será el encargado de llevar a cabo el registro de servicios. La máquina que sale elegida del proceso se autodenomina *Local Master Browser* (*LMB*) y se identifica mediante un nombre especial además del suyo propio. Su trabajo será mantener una lista de servicios que es el que acostumbramos a ver cuando utilizamos “Mis sitios de red” de Microsoft Windows.

Además de lo anterior, existe la figura del *DMB* (*Domain Master Browser*) que coordina las listas de servicios a través de distintos dominios NT, incluso a través de redes distintas. Utilizando el servicio *NBNS*, un *LMB* busca a su *DMB* e intercambia información con él. Actualmente, el mecanismo de sincronización hace que sea necesario bastante tiempo para que toda la información se propague por las distintas redes y aparezca de forma correcta en “Mis sitios de red”.

13.2. Sistema de archivos en red de *Sun Microsystems*: *NFS*

El sistema de archivos en red de *Sun Microsystems* (*NFS*) ha sido adoptado ampliamente en la industria y en entornos académicos desde su introducción en 1985. El diseño y desarrollo de *NFS* fueron emprendidos por el personal de *Sun Microsystems* en 1984. Aunque ya habían sido desarrollados varios servicios de archivos distribuidos, y utilizados con éxito, en universidades y laboratorios de investigación, *NFS* fue el primer servicio de archivos que fue diseñado como un producto. El diseño e implementación de *NFS* ha obtenido un éxito considerable tanto técnica como comercialmente.

Para animar su adopción como un estándar, las definiciones de las interfaces fundamentales fueron situadas en el dominio público, permitiendo a otros vendedores producir implementaciones, y el código fuente fue puesto disponible para una implementación de referencia a otros vendedores de computadores bajo licencia. Actualmente esta soportado por muchos vendedores y el protocolo *NFS* (versión 3) es un estándar de Internet, definido en la RFC 1813.

NFS proporciona acceso transparente a archivos remotos desde programas cliente ejecutándose sobre Linux y otros sistemas. Normalmente, cada computador tiene un cliente *NFS*

y módulos servidor instalados en el núcleo del sistema, al menos en el caso de los sistemas Linux. La relación cliente-servidor es simétrica: Cada computador en una red *NFS* puede actuar tanto como cliente como servidor, y los archivos en cada máquina pueden hacerse disponibles para acceso remoto desde otras máquinas. Cualquier computador puede ser un servidor, exportando algunos de sus archivos, y un cliente, accediendo a archivos de otras máquinas. Pero es una práctica habitual configurar instalaciones grandes con algunas máquinas como servidores dedicados y otras como estaciones de trabajo.

Un objetivo importante de *NFS* es conseguir un elevado nivel de soporte para la heterogeneidad de hardware y el sistema operativo. El diseño es independiente del sistema operativo gracias a la utilización de *XDR* (*eXternal Data Representation*): Existen implementaciones de servidor y cliente para casi todos los sistemas operativos y plataformas actuales, incluyendo Windows 95, Windows NT, MacOS, NetWare, OS/2 y VMS así como Linux y casi cualquier otra versión de Unix. Se han desarrollado implementaciones de *NFS* en máquinas multiprocesador de altas prestaciones por varios vendedores y éstas se utilizan ampliamente para satisfacer los requisitos de almacenamiento en intranets con muchos usuarios concurrentes.

13.2.1. Beneficios proporcionados por *NFS*

Algunos de los beneficios más destacados que *NFS* proporciona son:

- Las estaciones de trabajo locales utilizan menos espacio de disco debido a que los datos se encuentran centralizados en un único lugar pero pueden ser accedidos y modificados por varios usuarios, de tal forma que no es necesario replicar la información.
- Los usuarios no necesitan disponer de un directorio */home* en cada una de las máquinas cliente. Los directorios */home* pueden crearse en el servidor de *NFS* para posteriormente poder acceder a ellos desde cualquier máquina a través de la infraestructura de red.
- También se pueden compartir a través de la red dispositivos de almacenamiento como disquetes, CD-ROM y unidades ZIP. Esto puede reducir la inversión en dichos dispositivos y mejorar el aprovechamiento del hardware existente.

13.3. Sistema de información de redes de *Sun Microsystems*: *NIS*

El sistema de información de redes (*NIS*) es un servicio para redes que sirve para compartir información de sistemas. Los datos que normalmente administra *NIS* incluyen: contraseñas de usuario e información de grupos, asignaciones de *hosts* a dirección IP, alias de correo y otros datos de sistema que deben ser comunes entre los sistemas. Al centralizar la administración de estos datos, *NIS* permite que el administrador de sistemas cree un entorno más homogéneo, y a la vez simplifica la tarea de actualizar y distribuir estos datos.

NIS fue creado por *Sun Microsystems* para *SunOS*. Originariamente se conocía como *Yellow Pages* (*YP*), pero se le tuvo que cambiar el nombre como resultado de un procedimiento judicial instado por *British Telecom*. Oficialmente lo rebautizaron como *NIS*, pero quedaron vestigios de las convenciones de nombres *YP* del sistema. *NIS* se implementa por medio del protocolo para redes *ONDRPC* (Llamada a procedimientos remotos para computación en redes).

Los conjuntos de datos de *NIS* están representados como una o más asignaciones. Las asignaciones *NIS* están formadas por pares clave-valor organizadas de acuerdo con la forma en que las distintas funciones de bibliotecas de sistemas esperan acceder a los datos. Por ejemplo, dos asignaciones representan información del nombre de *host* en *NIS*: *hosts.byname* y *hosts.byaddr*. Estas asignaciones son un reflejo directo de cómo las funciones de biblioteca del

sistema *gethostbyname()* y *gethostbyaddress()* da una dirección IP, mientras que dada una dirección IP, *gethostbyaddress()* devuelve un nombre de host.

Todos los sistemas que comparten la misma base de datos *NIS* forman parte del mismo dominio *NIS*. Dentro de un dominio, un sistema puede tener uno o tres papeles: cliente, servidor esclavo o servidor maestro.

- **Maestro:** El sistema es el origen de los datos *NIS* para el dominio. Todas las actualizaciones de los datos *NIS* se producen en este *host*.
- **Esclavo:** El sistema mantiene una copia de la base de datos maestra. Se actualiza cuando los datos cambian en el maestro.
- **Cliente:** El sistema se conecta a un esclavo *NIS* o servidor maestro para obtener datos *NIS*.

Los dominios *NIS* difieren mucho de los dominios *DNS*: Mientras que los dominios *DNS* especifican los nodos de un gráfico, los dominios *NIS* son entidades singulares. No existe relación jerárquica entre los dos dominios *NIS*, incluso si una convención de nombres pueda sugerir que sí la hay.

13.4. Integrando *NIS* y *NFS*

Muchas veces los usuarios trabajan en computadores diferentes, teniendo que ingresar el usuario y contraseña que tienen creados en cada uno de esos equipos, y muchas veces también tienen que ir haciendo copias de sus trabajos en todos esos equipos. Claramente, esto trae consigo un gran problema, entonces, ¿Porque no mantener los archivos de cada usuario en un solo lugar y que cada vez que los usuarios se conecten a cualquier computadora estos puedan trabajar con sus datos como si estuvieran en su máquina local?, ¿Porque no mantener la información de su cuenta en un solo lugar, permitiéndoles con esto trabajar en cualquier computadora utilizando una única cuenta y clave?

Pues con la integración *NIS* y *NFS* podemos lograr esto. Así es, con *NIS* podemos hacer que el usuario crea que se está autenticando en la computadora local sin saber que realmente lo está haciendo en un servidor que mantiene centralizada la información de las cuentas de usuario. Con *NFS* podemos montar los datos del usuario que están en el servidor sobre la computadora local. Todo esto de manera transparente.

NIS se encarga de resolver los siguientes problemas:

- Centraliza archivos de configuración replicados como el */etc/passwd* en una sola máquina.
- Elimina las copias duplicadas de usuarios e información del sistema, permitiéndole al administrador hacer cambios en un solo sitio.

NFS resuelve los siguientes problemas:

- Hace parecer a los sistemas de archivos remotos como si fueran locales ya que oculta su verdadera ubicación física.
- Un usuario puede ver sus archivos, independientemente de donde estén localizados, ya sea que estén en el disco local, en un disco compartido, en un servidor o en una máquina que está al otro lado de una WAN.

NIS y *NFS* se complementan, ya que el exportar un sistema de archivos a una máquina en donde estos no existen violaría las reglas de integridad y seguridad impuestas.

NIS y *NFS* utilizan protocolos de redes para comunicarse con otras máquinas en la red.

NIS y *NFS* trabajan utilizando el modelo cliente-servidor. Básicamente, un cliente es una entidad que solicita un servicio a un servidor y un servidor es la entidad que provee el recurso solicitado por el cliente. En este esquema, *RPC* (*Remote Procedure Call*) juega un papel importante ya que *NIS* y *NFS* se basan en sus servicios. En vez de ejecutar el procedimiento en una máquina local, *RPC* pasa una serie de argumentos al procedimiento en un datagrama de red. El cliente *RPC* crea la sesión al localizar al servidor apropiado y se envía el datagrama a un proceso en el servidor que puede ejecutar la llamada del procedimiento remoto. En el servidor el argumento es desempacutado, el servidor ejecuta el comando y retorna las respuestas, si hay alguna, al cliente. De vuelta en el cliente, el valor del *RPC* es convertido a un valor esperado por la función que lo llamo y la aplicación continúa como si un procedimiento local lo hubiera sido llamado. La ubicación de puertos para una sesión es manejada por el demonio llamado *portmap*.

TEMA 7: SISTEMAS DE RED

Objetivos

- Estudiar los conceptos de TCP/IP necesarios para administrar y configurar un sistema Linux.
- Estudiar y diferenciar los archivos de configuración de red más importantes en un sistema Linux.
- Describir algunas aplicaciones seguras para administrar sistemas Linux de forma remota.

Contenido

1. Introducción a TCP/IP
2. Servicios sobre TCP/IP
3. ¿Qué es TCP/IP?
4. La pila de protocolos IP
5. Dispositivos físicos (hardware) de red
6. Conceptos TCP/IP
7. Direcciones TCP/IP
8. Componentes de la dirección de red
 - 8.1. Dirección de la máscara de red
 - 8.2. Dirección de red
 - 8.3. Dirección de puerta de acceso
 - 8.4. Dirección de difusión
9. Configurar la red
 - 9.1. Configuración de la interfaz *NIC* (*Network Interface Controller*)
 - 9.2. Configuración del *Name Resolver*
 - 9.3. Configuración del encaminamiento
 - 9.4. Configuración del *inetd*
 - 9.5. Configuración adicional: */etc/protocols* y */etc/networks*
 - 9.6. Algunos aspectos de seguridad a tomar en cuenta
10. Aplicaciones seguras
 - 10.1. *Secure SHell* (*ssh*)
 - 10.1.1. Características del protocolo *ssh*
 - 10.1.2. ¿Por qué usar *ssh*?
 - 10.1.3. Ejemplos de uso
 - 10.2. *Secure CoPy* (*scp*)
 - 10.2.1. El Protocolo *scp*
 - 10.2.2. La aplicación *scp*
 - 10.2.3. Ejemplos de uso

Bibliografía

Básica

- Josep Jorba Esteve, Remo Suppi Boldrito, “Administración avanzada de GNU/Linux Primera Edición”, UOC Formación de Posgrado, Software libre, 2004.
<http://www.uoc.edu/masters/esp/img/871.pdf>

Complementaria

- Dee-Ann LeBlanc, “Administración de sistemas LINUX La biblia”. Editorial ANAYA MULTIMEDIA, 2001.

<http://www.uoc.edu/masters/esp/img/871.pdf>

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.
- M. Carling, Stephen Degler, James Dennis, “Administración de Sistemas Linux, Guía Avanzada”, Editorial Prentice Hall, 2000.
- Iñaki Alegría Loinaz, Roberto Cortiñas Rodríguez, Aitzol Ezeiza Ramos, “Linux Administración del sistema y la red”, Editorial Prentice Hall, 2005.

La gestión de la red y el análisis del tráfico son dos tareas primordiales en la administración de sistemas Linux. En este sentido, la configuración de la red es un proceso relevante que se lleva a cabo configurando correctamente las tarjetas de red y las tablas de encaminamiento de todas y cada una de las máquinas dentro de la red.

1. Introducción a TCP/IP

El protocolo TCP/IP sintetiza un ejemplo de estandarización y una voluntad de comunicación a nivel global. El protocolo es en realidad un conjunto de protocolos básicos que se han ido agregando al principal para satisfacer las diferentes necesidades en la comunicación ordenador-ordenador. Dentro del conjunto de protocolos básicos agregados al protocolo principal podemos mencionar a: TCP, UDP, IP, ICMP, ARP, etc.

Algunas de las utilizaciones más frecuentes de TCP/IP por parte de los usuarios en la actualidad son las conexiones remotas a otros ordenadores mediante *telnet* (*TELEcommunication NETwork*) o *ssh* (*Secure SHell*), la utilización de ficheros remotos mediante *NFS* (*Network File System*) o la transferencia de ficheros mediante *ftp* (*File Transfer Protocol*), *scp* (*Secure CoPy*) o *http* (*HiperText Transfer Protocol*).

2. Servicios sobre TCP/IP

Dentro de los servicios TCP/IP tradicionales más importantes que podemos encontrar tenemos:

- **Transferencia de archivos:** Mediante el protocolo *ftp* podemos transferir archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor, de manera que un usuario desde un equipo cliente puede conectarse a un servidor para descargar archivos desde él o para enviarle sus propios archivos independientemente del sistema operativo utilizado en cada equipo. Para ello, el usuario deberá tener una cuenta de acceso en el servidor e identificarse a través de su nombre y su contraseña; o puede conectarse a servidores donde existen repositorios de información, tales como software, documentación, videos, manuales, etc., en donde el usuario se conectará de forma anónima para transferir estos archivos a su ordenador local.
- **Conexión remota:** El protocolo de red *telnet* permite a un usuario conectarse a un ordenador de forma remota. El ordenador local se utiliza como terminal del ordenador remoto y todo es ejecutado sobre éste permaneciendo el ordenador local invisible desde el punto de vista de la sesión. El mayor inconveniente que presenta *telnet* esta relacionado con aspectos de seguridad, ya que todos los nombres de usuarios y contraseñas necesarias para entrar en las máquinas remotas viajan por la red como texto plano, es decir cadenas de texto sin cifrar. Esto facilita que cualquiera que espíe el tráfico de la red pueda obtener los nombres de usuario y contraseñas, y de esa manera él también poder acceder a todas esas máquinas. Por esta razón *telnet* dejó de usarse casi totalmente, hace unos años, cuando apareció y se popularizó *ssh*, que puede describirse como una versión cifrada de *telnet*, cabe destacar que lograr cifrar las cadenas de texto que viajan por la red involucra un coste añadido a la comunicación.
- **e-mail:** Este servicio permite enviar mensajes a los usuarios de otros ordenadores. Este modo de comunicación se ha transformado en un elemento vital en la vida de los usuarios. En la actualidad los e-mails son enviados a un servidor central para que después los usuarios puedan recuperarlos por medio de programas específicos o por medio de lecturas de los mismos a través de una conexión web.

El avance de la tecnología y el bajo coste de los ordenadores han permitido que determinados servicios se hayan especializado en ello y se ofrecen configurados sobre

determinados ordenadores trabajando en un modelo cliente-servidor. Un servidor es un sistema que ofrece un servicio específico para el resto de ordenadores existentes en la red. Un cliente es otro ordenador que utiliza este servicio. Todos estos servicios generalmente son ofrecidos bajo TCP/IP.

Otros servicios tradicionales que son ofrecidos bajo TCP/IP y que gran parte de sus implementaciones giran en torno a la arquitectura cliente-servidor son:

- **Sistemas de archivos en red:** Permite a un sistema acceder a los archivos sobre un sistema remoto en una forma más integrada que cuando se hace utilizando *ftp*. Los dispositivos de almacenamiento, o parte de ellos, son exportados hacia el sistema que desea acceder y éste los puede utilizar como si fueran dispositivos locales.
- **Impresión remota:** Permite llevar a cabo tareas de impresión desde impresoras conectadas en ordenadores remotos.
- **Ejecución remota:** Permite que un usuario ejecute un programa sobre otro ordenador. Existen diferentes maneras de realizar esta ejecución: la primera es a través de la ejecución de un comando, como por ejemplo: *rsh*, *ssh*, *rexec*, etc., y la segunda es a través de sistemas con mecanismos *RPC* (**R**emote **P**rocedure **C**all), dichos mecanismos permiten a un programa en un ordenador local ejecutar una función de un programa ubicada sobre un ordenador remoto. Existen diversas implementaciones de los mecanismos *RPC*, pero las más comunes son *Xerox's Courier* y *Sun's RPC*, esta última adoptada por la mayoría de los sistemas Linux.
- **Centralización de datos:** Generalmente en las grandes instalaciones donde existen muchos sistemas de cómputo, existe un gran conjunto de datos que necesitan ser centralizados para mejorar su utilización, por ejemplo, nombres de usuarios, contraseñas de usuarios, direcciones de red, etc. Todo ello facilita que un usuario disponga de una única cuenta y por ende poder acceder con su misma cuenta a cualquier máquina perteneciente a la organización. Por ejemplo, *NIS* está diseñado para manejar todo este tipo de datos y se encuentra disponible para la mayoría de sistemas Linux.
- **Servidores de terminal:** Permite conectar terminales a un servidor que ejecuta *telnet* para conectarse a un ordenador central. Este tipo de instalaciones permite básicamente reducir costes y mejorar las conexiones al ordenador central.
- **Servidores de terminales gráficas:** Permiten que un ordenador pueda visualizar información gráfica sobre un *display* que está conectado a un ordenador remoto. El más común de estos sistemas es *X Window*.

3. ¿Qué es TCP/IP?

TCP/IP son en realidad dos protocolos de comunicación entre ordenadores que son independientes el uno del otro.

Por un lado, TCP es un protocolo de nivel de transporte (ver figura 7.1) que define las reglas de comunicación para que un *host* pueda llevar a cabo una comunicación con otro *host* remoto. TCP es un protocolo fiable: porque se encarga de llevar a cabo retransmisiones en caso de pérdidas de segmentos; también se encarga de ordenar los segmentos en el destino y además descarta segmentos que hayan podido llegar duplicados, es un protocolo seguro: porque utiliza un mecanismo de preguntas y respuestas (*ACK* y *NACK*) para saber si los segmentos han llegado correctamente al destino y es un protocolo orientado a conexión: porque primero establece una conexión con el destino, con el objetivo de negociar parámetros sobre la

comunicación para que una vez que se hayan establecido dichos parámetros enviar los datos. Cabe destacar que en TCP la comunicación se trata como un flujo de datos (*stream*).

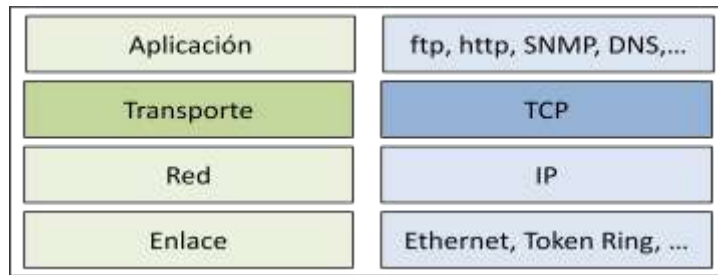


Figura 7.1: Ubicación de TCP en la pila de protocolos

Por otro lado, IP es un protocolo de nivel de red (ver figura 7.2) que permite identificar las redes y establecer los caminos entre los diferentes ordenadores. IP encamina los datos entre dos ordenadores a través de las redes. Además, IP provee un servicio de datagramas no fiable, ya que no provee ningún mecanismo para determinar si un paquete alcanza o no su destino y únicamente proporciona seguridad de sus cabeceras, mediante sumas de comprobación de las mismas. Es por esto que la fiabilidad es proporcionada por protocolos de la capa de transporte como *TCP*.

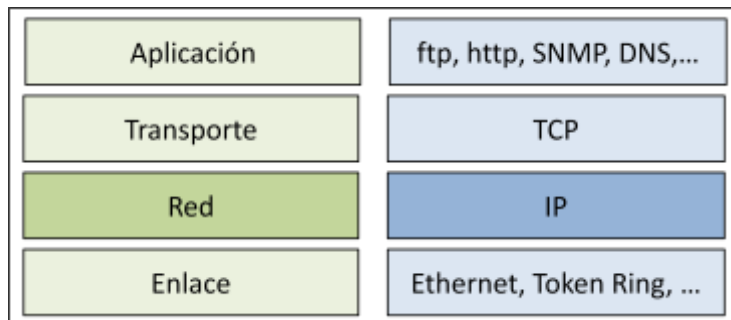


Figura 7.2: Ubicación de IP en la pila de protocolos

Una alternativa de implementación de un protocolo más ligero que TCP la conforma el protocolo UDP (*User Datagram Protocol*) el cual es también un protocolo de nivel de transporte (ver figura 7.3) basado en el intercambio de datagramas. Permite el envío de los datagramas a través de la red sin que se haya establecido previamente una conexión, razón por la cual es no orientado a la conexión. Es un protocolo no fiable: ya que no sabe si los datagramas han llegado correctamente al destino y por lo tanto no se preocupa por llevar a cabo retransmisiones de datagramas. El protocolo UDP tiene la ventaja de que ejerce una menor sobrecarga a la red que las conexiones con TCP, pero presenta los inconvenientes mencionados anteriormente.

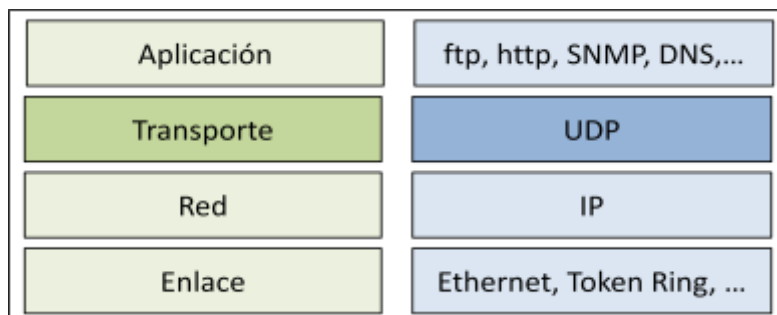


Figura 7.3: Ubicación de UDP en la pila de protocolos

Existe un sub-protocolo de control y notificación de errores del protocolo IP, este sub-protocolo es conocido como ICMP (*Internet Control Message Protocol*). ICMP se utiliza para enviar mensajes de error, indicando por ejemplo que un servicio determinado no está disponible o que un *router* o *host* no puede ser alcanzado. Los mensajes ICMP son construidos en el nivel de capa de red y su principal utilidad es la de manejar mensajes de error y de control necesarios para los sistemas de la red, informando con ellos a la fuente original para que evite o corrija el problema detectado.

En resumen, TCP/IP es una familia de protocolos, que incluye a: IP, TCP, UDP, ICMP, etc., que proveen un conjunto de funciones a bajo nivel utilizadas por la mayoría de las aplicaciones.

Existe actualmente una nueva versión del protocolo IPv4, conocido como IPv6 o también llamado IPng (*IP next generation*) que mejora notablemente al protocolo IPv4 en temas tales como mayor número de *hosts*, control de tráfico, seguridad o mejoras en aspectos de encaminamiento.

4. La pila de protocolos IP

Durante la etapa de diseño, los creadores de IPv4 decidieron aislar las aplicaciones computarizadas de los tipos diferentes de hardware de redes que existían y que previeron que existirían en un futuro no tan lejano. También previeron que la mejor manera de alentar la implantación de unas comunicaciones fiables se podía hacer a través de la integración de un nivel de comunicaciones fiable del paquete IP. Con esto en sus mentes, construyeron un planteamiento de niveles conceptuales al que denominaron pila de protocolos IP.

Los protocolos están ordenados en niveles con el fin de facilitar la opción de compartir hardware para redes entre aplicaciones que tengan requerimientos divergentes. Esto permite que los protocolos de nivel de aplicación difieran a la vez que confían en protocolos comunes de nivel inferior. La figura 7.4 muestra los protocolos ordenados de la pila IP.

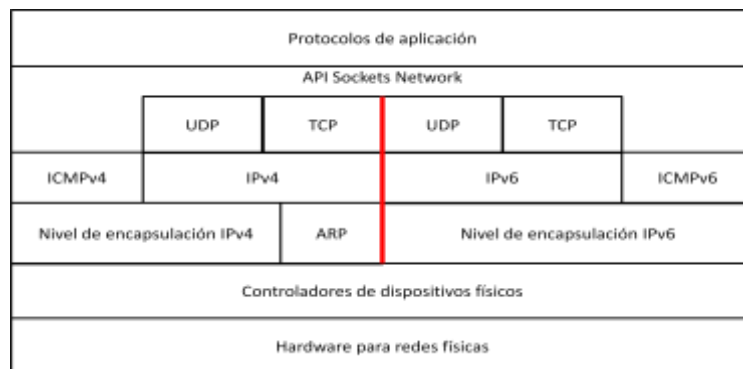


Figura 7.4: Implementación de niveles de redes

5. Dispositivos físicos (hardware) de red

Desde el punto de vista físico, el hardware más utilizado para LAN es conocido como Ethernet, aunque hoy en día existen tecnologías que son más veloces como: FastEthernet y EthernetGigabit. Dentro de las ventajas más destacadas de Ethernet están: su bajo coste, sus velocidades aceptables que pueden ser de 10, 100, o 1,000 megabits por segundo y la facilidad en su instalación.

Los Ethernet se pueden clasificar en tres tipos: gruesos, finos, y de par trenzado. Los dos primeros pueden usar cable coaxial, estos difirieren en el grosor y el modo de conectar este cable a los *hosts*. El Ethernet fino emplea conectores *BNC* (*Bayonet Neill-Concelman*) con forma de T, que se pinchan en el cable y se enganchan a los conectores de la parte trasera del

ordenador. El Ethernet grueso requiere que se realice un pequeño agujero en el cable, y se conecte un transceptor utilizando un *conector vampiro* luego, se podrán conectar uno o más *hosts* al transceptor. Los cables Ethernet fino y grueso pueden alcanzar una distancia de 200 y 500 metros respectivamente. Estos también son conocidos como 10base-2 y 10base-5 respectivamente. Los cables Ethernet fino y grueso están actualmente obsoletos y han sido reemplazados por el par trenzado el cual utiliza cables hechos de hilos de cobre entrelazados entre si. La conexión por par trenzado es conocida como 10baseT o 100baseT según sea la velocidad.

La tecnología Ethernet utiliza elementos intermedios de comunicación como pueden ser: *hubs*, *switchs* o *routers*, los cuales permiten configurar múltiples segmentos de red y dividir el tráfico para mejorar las prestaciones de transferencia de información. Normalmente, en las grandes instituciones estas LAN Ethernet están interconectadas a través de fibra óptica utilizando tecnología FDDI (*Fiber Distributed Data Interface*) que es mucho más cara y compleja de instalar, pero que permite obtener velocidades de transmisión superiores a las alcanzadas con Ethernet además no presenta las limitaciones de distancia que presenta Ethernet ya que FDDI admite distancias de hasta 200 km. Su coste se justifica para enlaces entre edificios o entre segmentos de red muy congestionados.

Existe además otro tipo de hardware menos común, pero no menos interesante como es ATM (*Asynchronous Transfer Mode*). Este hardware permite montar LAN con una calidad de servicio elevada y es una buena opción cuando deben montarse redes de alta velocidad y baja latencia, como por ejemplo aquellas que involucren distribución de vídeo en tiempo real.

Existen otros hardware soportados por GNU/Linux para la interconexión de ordenadores, entre los cuales podemos mencionar: Frame Relay o X.25, utilizados en ordenadores que acceden o interconectan WAN y en servidores con grandes necesidades de transferencias de datos. Otro hardware que merece una mención es Packet Radio, utilizado para interconexión vía radio utilizando protocolos como AX.25, NetRom o Rose. Además de los mencionados anteriormente los sistemas GNU/Linux pueden hacer uso de dispositivos dialing up, que utilizan líneas series, lentas pero muy baratas, a través de módems analógicos o digitales como pueden ser: RDSI, DSL, ADSL, etc. Estas últimas son las que comúnmente se utilizan en pymes o uso doméstico y requieren otro protocolo para la transmisión de paquetes, tal como SLIP (*Serial Line Internet Protocol*) o PPP (*Point-to-Point Protocol*).

Para virtualizar la diversidad de hardware sobre una red, TCP/IP define una interfaz abstracta mediante la cual se concentrarán todos los paquetes que serán enviados por un dispositivo físico, lo cual también significa una red o un segmento de esta red. Por ello, por cada dispositivo de comunicación en la máquina tendremos una interfaz correspondiente en el *kernel* del sistema operativo.

Ethernet en GNU/Linux se llaman con **ethX** (donde en todas, **X** indica un número de orden comenzando por 0), la interfaz de líneas series (módems) se llaman por **pppX** (para PPP) o **slX** (para SLIP), para FDDI son **fdiX**. Estos nombres son utilizados por los comandos para configurar las interfaces y asignarles el número de identificación que posteriormente permitirá comunicarse con otros dispositivos en la red.

Los dispositivos de red pueden ser listados en el directorio */dev* que es donde existe un archivo, que representa a cada dispositivo hardware, ya sea de modo bloque o de modo carácter según sea su transferencia.

Para poder ver las interfaces de red disponibles en nuestro sistema podemos hacer uso del comando *ifconfig* junto con la opción *-a*.

Ejemplo:

```
$> ifconfig -a
eth0 Link encap:Ethernet HWaddr 00:E0:B8:EB:08:32
      inet dirección:172.16.118.9 Bcast:172.16.255.255 Máscara:255.255.0.0
      dirección inet6: fe80::2e0:b8ff:feeb:832/64 Alcance:Vínculo
      ARRIBA BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:172 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:0 (0.0 b) TX bytes:13240 (12.9 KiB)

lo    Link encap:Bucle local
      inet dirección:127.0.0.1 Máscara:255.0.0.0
      dirección inet6: ::1/128 Alcance:Anfitrión
      ARRIBA LOOPBACK RUNNING MTU:16436 Metric:1
      RX packets:42 errors:0 dropped:0 overruns:0 frame:0
      TX packets:42 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:3904 (3.8 KiB) TX bytes:3904 (3.8 KiB)
```

6. Conceptos TCP/IP

Según lo expuesto en las secciones anteriores, la comunicación involucra un sin número de términos y conceptos que serán ampliados a continuación.

Intranet: El término intranet se refiere a la aplicación de tecnologías de Internet dentro de una organización básicamente para distribuir y tener disponible información dentro de la compañía, sin hacer uso del propio Internet. Por ejemplo, los servicios ofrecidos por GNU/Linux como servicios tanto para Internet como para intranet incluyen: correo electrónico, www, news, etc.

Host: Se denomina *host* a una máquina que se conecta a la red. En un sentido amplio un *host* puede ser un ordenador, una impresora, una torre (*rack*) de CD, etc., es decir, un elemento activo y diferenciable en la red que reclama o presta algún servicio y/o comparte información.

Dirección de red Ethernet: La dirección MAC (*Media Access Control Address*) es un identificador de 48 bits que corresponde de forma única a una tarjeta o interfaz de red. Cada dispositivo tiene su propia dirección MAC determinada y configurada por el IEEE, los últimos 24 bits, y por el fabricante, los primeros 24 bits.

Las direcciones MAC son únicas a nivel mundial, puesto que son escritas directamente, en forma binaria, en el dispositivo físico (hardware) del controlador de red Ethernet en su momento de fabricación.

Host name: Cada *host* debe tener además un único nombre en la red. Ellos pueden ser sólo nombres o bien utilizar un esquema de nombres jerárquico basado en dominios (*hierarchical domain naming scheme*). Los nombres de los *hosts* deben ser únicos, lo cual resulta fácil en pequeñas redes, pero se vuelve más dificultoso en redes extensas e imposible en Internet si no se realiza algún control. Los nombres deben ser de un máximo de 32 caracteres entre a-zA-Z0-9-., y que no contengan espacios o # comenzando por un carácter alfabético.

En Linux podemos averiguar el nombre de la máquina mediante el comando *hostname*.

Dirección de Internet: Una dirección IP es un número de 32 bits que identifica de manera lógica y jerárquica a un interfaz de un dispositivo, habitualmente una computadora, dentro de una red que utilice el protocolo IP. Está compuesta por cuatro números en el rango 0-255 separados por puntos, por ejemplo 192.168.1.1.

Cuando nos conectamos en algún sitio para acceder a Internet es necesario utilizar una dirección IP. Esta dirección IP puede cambiar al reconectarnos; a esta forma de asignación de dirección IP se le denomina dirección IP dinámica o IP dinámica.

Los sitios de Internet que por su naturaleza necesitan estar permanentemente conectados, generalmente tienen una dirección IP fija o IP estática, es decir, que la dirección IP no cambia con el tiempo. Los servidores de correo, servidores web, DNS, *ftp* públicos, etc. necesariamente deben contar con una dirección IP fija o estática, ya que de esta forma se permite su localización en la red.

A través de Internet, los ordenadores se conectan entre sí mediante sus respectivas direcciones IP. Sin embargo, a los seres humanos nos es más cómodo utilizar otra notación más fácil de recordar y utilizar, como los nombres de dominio; la traducción entre unos y otros se resuelve mediante los servidores de nombres de dominio DNS.

Puerto: Es un identificador numérico del *buzón* en un *host* que permite que un mensaje TCP o UDP pueda ser leído por una aplicación concreta dentro de este *host*. También son conocidos como las direcciones de las aplicaciones o como SAP (*Service Access Point*). Tanto a nivel de UDP como de TCP se repiten los puertos y estos están comprendidos entre 0 y 65535 (16 bits). Por ejemplo, dos máquinas que se comuniquen por *telnet* lo harán por el puerto 23, pero las dos mismas máquinas pueden tener otra comunicación al mismo tiempo vía *ftp* por el puerto 21. Esto nos lleva a que es posible tener diferentes aplicaciones comunicándose entre dos *host* a través de diferentes puertos simultáneamente.

Router: Es un dispositivo de hardware que opera en el nivel de red y que nos permite realizar interconexiones entre redes de computadoras. Este dispositivo permite asegurar el enrutamiento de paquetes entre redes o determinar la ruta que debe tomar el paquete de datos. Según sean sus características, podrá transferir información entre dos redes de protocolos similares o diferentes. Las máquinas Linux pueden ser configuradas para hacer de *router*, aunque también existen equipos especializados para esta tarea y es lo que comúnmente se suele utilizar.

Encaminamiento: Representa una de las funciones que un *router* es capaz de llevar a cabo. Esta función se refiere al modo en que los mensajes son enviados a través de distintas redes o subredes. Para esta labor los *routers* utilizan una tabla para hacer el encaminamiento de los paquetes, esta tabla es conocida con el nombre de tabla de encaminamiento, en ella existe una entrada por cada una de las redes o subredes a las cuales el *router* sabe llegar, además existe una entrada por defecto la cual es especialmente útil cuando dada una dirección IP, a la cual queremos enviar un paquete, no existe una entrada en la tabla de encaminamiento para llegar hacia la red de destino que contiene la IP a la cual queremos enviar el paquete, entonces lo que se suele hacer es utilizar la entrada por defecto, la cual hará saltar el paquete a la dirección IP que corresponda al *gateway* por defecto.

Cabe destacar que no solo los *routers* poseen tablas de encaminamiento, ya que también los *hosts* contienen sus propias tablas. Para conocer la tabla de encaminamiento de un ordenador Linux podemos hacer uso del comando *netstat* con la opción *-rn*.

DNS (*Domain Name System*): Es una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet. Aunque como base de datos el DNS es capaz de asociar diferentes tipos de información a cada nombre, los usos más comunes son la asignación de nombres de dominio a direcciones IP y la localización de los servidores de correo electrónico de cada dominio.

La asignación de nombres a direcciones IP es ciertamente la función más conocida de los DNS. Por ejemplo, si la dirección IP del sitio *ftp* de *unanleon.edu.ni* es 200.64.128.4, la mayoría de la gente llega a este equipo especificando *ftp.unanleon.edu.ni* y no la dirección IP. Además de ser más fácil de recordar, el nombre es más fiable ya que la dirección IP podría cambiar por muchas razones, sin que tenga que cambiar el nombre.

DHCP (*Dynamic Host Configuration Protocol*): Es un protocolo de red que permite a los *hosts* de una red IP obtener sus parámetros de configuración automáticamente. Se trata de un protocolo de tipo cliente-servidor en el que generalmente un servidor posee una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas van estando libres, sabiendo en todo momento quién ha estado en posesión de esa IP, cuánto tiempo la ha tenido y a quién se la ha asignado después. Provee los parámetros de configuración a las computadoras conectadas a la red informática con la pila de protocolos TCP/IP, como por ejemplo máscara de red, puerta de enlace y otros.

Sin *DHCP*, cada dirección IP debe configurarse manualmente en cada computadora y, si la computadora es reubicada en otra parte de la red, se debe configurar otra dirección IP diferente. *DHCP* permite al administrador supervisar y distribuir de forma centralizada las direcciones IP necesarias y, automáticamente, asignar y enviar una nueva IP si la computadora es conectada en un lugar diferente de la red.

El protocolo *DHCP* incluye tres métodos de asignación de direcciones IP:

- **Asignación manual o estática:** Asigna una dirección IP a una máquina determinada. Útil cuando queremos controlar la asignación de dirección IP a cada cliente, y evitar, también, que se conecten clientes no identificados.
- **Asignación automática:** Asigna una dirección IP de forma permanente a una máquina cliente la primera vez que hace la solicitud al servidor *DHCP* y hasta que el cliente la libera. Útil cuando el número de clientes no varía demasiado.
- **Asignación dinámica:** el único método que permite la reutilización dinámica de las direcciones IP. El administrador de la red determina un rango de direcciones IP y cada computadora conectada a la red está configurada para solicitar su dirección IP al servidor cuando la tarjeta de interfaz de red se inicializa. El procedimiento usa un concepto muy simple en un intervalo de tiempo controlable. Esto facilita la instalación de nuevas máquinas clientes a la red.

DHCP es una alternativa a otros protocolos de gestión de direcciones IP de red, como el *BOOTP (*BOOTstrap Protocol*)*. *DHCP* es un protocolo más avanzado, pero ambos son los usados normalmente.

ARP, RARP: En algunas redes, como por ejemplo 802.3 LAN (correspondiente al estándar de Ethernet), las direcciones IP son descubiertas automáticamente a través de dos protocolos miembros del *IPS (*Internet Protocol Suite*)*. Estos protocolos son el *ARP (*Address Resolution Protocol*)* y el *RARP (*Reverse Address Resolution Protocol*)*. *ARP* utiliza mensajes de tipo *broadcast* para determinar la dirección MAC correspondiente a una determinada dirección IP dentro de una red particular. *RARP* también utiliza mensajes de tipo *broadcast* para determinar una dirección IP dentro de una red particular asociada con una determinada dirección MAC.

Interfaz de sockets: Para comunicarse de forma libre, las aplicaciones Linux poseen una *API (Application Programming Interface)* sencilla que se conoce como interfaz de *sockets*. Esta *API* data de las primeras implementaciones para redes *BSD* y únicamente ha requerido pequeñas dosis de trabajo para satisfacer las necesidades de los programadores de redes en las últimas dos décadas. Para TCP/IP en Linux la *API* más común es la *Berkeley Socket Library*, esta permite crear un punto de comunicación, conocido como *socket*, luego asociar éste *socket* con una dirección y un puerto, mediante la llamada a la función *bind()*, y por último ofrecer el servicio de comunicación, a través de llamadas a funciones como: *connect()*, *listen()*, *accept()*, *send()*, *sendto()*, *recv()*, *recvfrom()*.

7. Direcciones TCP/IP

Cuando se tiene un conjunto de máquinas comunicadas entre ellas por medio de una red o un conjunto de redes, los protocolos de red necesitan un modo de determinar de qué máquina procede el mensaje y a qué máquina se dirige el mensaje. TCP/IP utiliza una dirección IP con este fin.

La dirección IP está compuesta por 32 bits y se representan en formato punto decimal; estas a su vez se dividen en dos campos de tamaño variable: la dirección de la red y la dirección del *host*. La dirección de la red está compuesta por los bits de orden superior y la dirección del *host* está compuesta por el resto de bits.

Cabe destacar que existen direcciones especiales que se utilizan en TCP/IP como la 0.0.0.0, la cual indica el propio *host*, la 255.255.255.255, la cual indica una difusión limitada a la propia red y la 127.*.*.*, la cual indica dirección(es) interna(s) de bucle local (*loopback*).

También a la hora de hacer las diferentes asignaciones de direcciones IP debemos saber que existen diferentes tipos de redes o direcciones, esto es conocido como direccionamiento por clases de red. El espacio de direcciones está dividido en cinco clases disjuntas:

Clase A (ver figura 7.5): Todas las direcciones de red de clase A empiezan con un 0 binario. Pero, como ya vimos antes las direcciones de red con el primer octeto puesto a 0 o que sean 127 están reservadas. Por lo tanto, existen 126 números de red potenciales de clase A en los cuales su primer octeto en formato punto decimal está en el rango de 1 a 126. El patrón binario es 0 + 7 bits para red + 24 bits para *hosts*. Y la máscara por defecto es: 255.0.0.0.

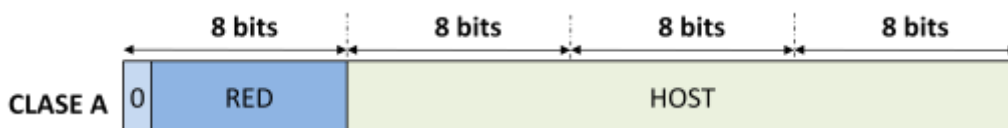


Figura 7.5: Direccionamiento para la clase A

Clase B (ver figura 7.6): Las direcciones de red de clase B comienzan con un número binario 10, de forma que su primer número decimal está entre 128 y 191. El segundo octeto también forma parte de la dirección de clase B, de forma que existen $2^{14} = 16.384$ direcciones de clase B. El patrón binario es 10 + 14 bits para red + 16 bits para *hosts*. Y la máscara por defecto es: 255.255.0.0.

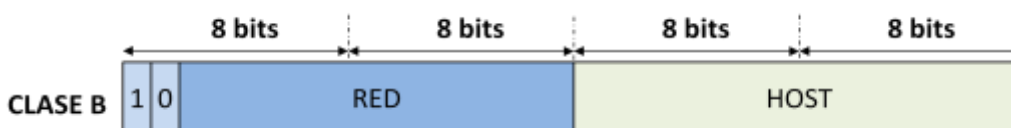


Figura 7.6: Direccionamiento para la clase B

Clase C (ver figura 7.7): Las direcciones de red de clase C comienzan con un número binario 110, de forma que su primer número decimal está entre 192 y 223. El tercer octeto también forma parte de la dirección de clase C, de forma que existen $2^{21} = 2.097.152$ direcciones de clase C. El patrón binario es 110 + 21 bits para red + 8 bits para *hosts*. Y la máscara por defecto es: 255.255.255.0.

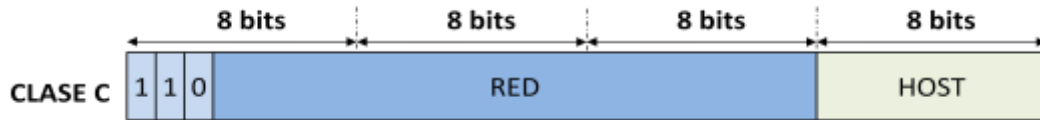


Figura 7.7: Direccionamiento para la clase C

Clase D (ver figura 7.8): Las direcciones de red de clase D comienzan con un número binario 1110, de forma que su primer número decimal está entre 224 y 239. Se utiliza para asignar direcciones de multidifusión o direcciones de *multicast*.

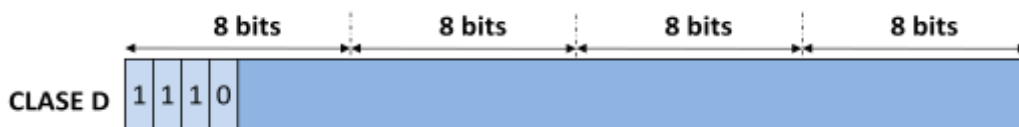


Figura 7.8: Direccionamiento para la clase D

Clase E (ver figura 7.9): Las direcciones de red de clase E comienzan con un número binario 11110, de forma que su primer número decimal está entre 240 y 255. Fue desarrollada como una clase para llevar a cabo investigaciones con el fin de encontrarle usos futuros.

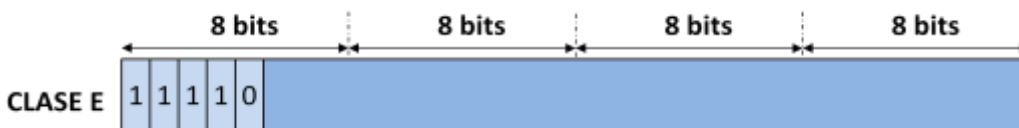


Figura 7.9: Direccionamiento para la clase E

Algunos rangos de direcciones han sido reservados para que no correspondan a redes públicas, sino a redes privadas, es decir, máquinas que se conectan entre ellas sin tener conexión con el exterior y por lo tanto los mensajes no serán encaminados a través de Internet, lo cual es comúnmente conocido como Intranet. Estas direcciones reservadas son para la clase A todas las direcciones de tipo 10.*.*, para la clase B van desde la 172.16.*.* hasta la 172.31.*.* y para la clase C van desde la 192.168.0.* hasta la 192.168.255.*.

8. Componentes de la dirección de red

Una LAN u otro tipo de redes tienen un grupo de direcciones especiales que se deben asignar para que todo funcione de forma adecuada. Algunas de estas direcciones son elegidas por el administrador, pero este debe conocer de ellas para hacer una correcta asignación de las mismas.

8.1. Dirección de la máscara de red

La primera dirección especial es la *máscara de red*. Por defecto, las *máscaras* de red de la tabla 7.1 se utilizan para las clases de dirección principales.

Clase	Máscara de red
Clase A	255.0.0.0
Clase B	255.255.0.0
Clase C	255.255.255.0

Tabla 7.1: Máscaras de red estándar

Un concepto muy importante que generalmente va de la mano con las máscaras de red es el concepto de subred. Subred significa subdividir la parte de *hosts* en pequeñas redes dentro de la misma red. Una subred toma la responsabilidad de enviar el tráfico a ciertos rangos de direcciones IP. El número de bits que son interpretados como identificadores de la subred es dado por la máscara de red (*netmask*) que esta representado por un número de 32 bits. Para obtener el identificador de la subred, podemos hacer una operación lógica AND entre la máscara de subred y la IP, lo cual dará la IP de la subred.

Mientras hablemos de una dirección de clase completa, sin tener en cuenta qué clase, estas máscaras de red se quedan tal y como están. Sin embargo, cuando queremos dividir una red en una o varias subredes, entonces tendremos que ajustar las máscara de red para reflejar ese echo.

8.2. Dirección de red

La dirección de red es la otra mitad de la ecuación que le indica a TCP/IP cómo es de grande una red junto con su máscara de red. Estos dos números puestos juntos en un formato binario deben ser iguales a cero. Normalmente, si tenemos una red de clase C, su dirección de red acaba en 0. Por ejemplo, la dirección de red para la máquina 192.168.15.12 en una LAN de clase C es 192.168.15.0.

8.3. Dirección de puerta de acceso

Una puerta de acceso es la interfaz que necesita el tráfico para viajar y alcanzar otra red, ya sea otra parte de la LAN o de Internet. Así que, por ejemplo, si tenemos una LAN sencilla con una máquina con un a tarjeta Ethernet gestionando la conexión a Internet, entonces esa dirección IP de la tarjeta Ethernet es la dirección de puerta de acceso.

8.4. Dirección de difusión

La dirección de difusión es una dirección especial, ya que cada *host* en una red escucha todos los mensajes, además de los que le corresponden a su propia dirección. Esta dirección permite que datagramas, generalmente de información de encaminamiento y de mensajes de aviso, puedan ser enviados a una red y todos los *hosts* del mismo segmento de red los puedan leer. Por ejemplo, cuando *ARP* busca encontrar la dirección Ethernet correspondiente a una determinada IP, éste utiliza un mensaje de en modo difusión, el cual es enviado a todas las máquinas de la red simultáneamente. Cada *host* en la red lee este mensaje y compara la IP que se busca con la propia y le retorna un mensaje al *host* que hizo la pregunta si hay coincidencia.

9. Configurar la red

En la siguiente sección analizaremos como se lleva a cabo la configuración de la red en un equipo Linux, además trabajaremos con algunos archivos de configuración de red, los cuales son secuencias de comandos de configuración de recursos de todo el sistema que se ejecutan en el momento del arranque.

9.1. Configuración de la interfaz *NIC* (*Network Interface Controller*)

Una vez que se encuentra cargado el *kernel* de GNU/Linux, éste ejecuta el comando *init* que a su vez lee el archivo de configuración */etc/inittab* y comienza el proceso de inicialización de los diferentes servicios, programas o registros que sean necesarios para que el sistema funcione como el usuario quiere o como el administrador lo ha establecido. Generalmente, el *inittab* contiene secuencias tales como: *si::sysinit:/etc/init.d/boot*, que representa el nombre del archivo de comandos (*script*) que controla las secuencias de inicialización. Generalmente este *script* llama a otros *scripts*, entre los cuales se encuentra el *script* de inicialización de la red.

En los sistemas Debian se ejecuta `/etc/init.d/network` para la configuración de la interfaz de red y en función del nivel de arranque; por ejemplo, en el nivel 2 se ejecutarán todos los ficheros S* del directorio `/etc/rc2.d`, que son enlaces al directorio `/etc/init.d`, y en el nivel de apagado se ejecutarán todos los ficheros K* del directorio `/etc/rc2.d`. De este modo, el *script* está sólo una vez y de acuerdo a los servicios deseados en ese estado se crea un enlace en el directorio correspondiente a la configuración del *host*-estado.

Los dispositivos de red se crean automáticamente cuando se inicializa el hardware correspondiente. Por ejemplo, el controlador de Ethernet crea las interfaces **ethX** secuencialmente cuando se localiza el hardware correspondiente.

A partir de este momento, se puede configurar la interfaz de red, lo cual implica dos pasos: asignar la dirección de red al dispositivo e inicializar los parámetros de la red al sistema. El comando utilizado para ello es *ifconfig* (**InterFace CONFIGure**).

Por ejemplo:

```
$> sudo ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

Lo cual indica configurar el dispositivo *eth0* asignándole la dirección IP 192.168.110.23 y la máscara de red 255.255.255.0. El *up* indica que la interfaz pasará al estado activo, si lo que se quisiera es desactivarla deberíamos ejecutar `sudo ifconfig eth0 down`. El comando asume que si algunos valores no se indican, estos tomarán los valores por defecto. En este caso, el *kernel* configurará esta máquina como una máquina con una dirección de clase C y configurará la dirección de *broadcast* con la dirección 192.168.110.255.

9.2. Configuración del *Name Resolver*

En esta sección configuraremos el *Name Resolver* que es el encargado de convertir nombres como `unanleon.edu.ni` en direcciones como `200.64.128.1`.

El archivo `/etc/resolv.conf` configura el sistema de resolución del nombre, especificando la dirección de un servidor de nombres, si existe alguno, y los dominios que en los cuales se desea buscar de forma predeterminada si un nombre de *host* no es un nombre de *host* especificado completamente. Su formato es muy simple, una línea de texto por cada sentencia. Existen tres palabras clave para tal fin: *domain*, *search* y *nameserver*.

La siguiente lista explica cada una de las palabras claves:

- **domain:** Aquí se debe especificar el dominio local.
- **search:** Aquí se debe especificar una lista de dominios alternativos.
- **nameserver:** Aquí se debe especificar la dirección IP del *Domain Name Server*.

Ejemplo de `/etc/resolv.conf`

```
$>cat /etc/resolv.conf
domain unanleon.edu.ni
search unanleon.edu.ni alternateunanleon.edu.ni
nameserver 200.64.128.4
nameserver 200.64.128.69
```

Un archivo importante es el */etc/host.conf*, que permite configurar el comportamiento del *Name Resolver*. Su importancia reside en indicar dónde se resuelve primero la dirección o el nombre de un *host*. Esta consulta puede ser realizada al servidor DNS o a tablas locales dentro de la máquina actual ubicadas en el archivo */etc/hosts*.

Ejemplo de */etc/host.conf*

```
$> cat /etc/host.conf
order hosts,bind
multi on
```

Esta configuración indica que primero se verifique el archivo */etc/hosts* antes de solicitar una petición al DNS y también indica, en la segunda línea, que retorne todas las direcciones válidas que se encuentren en */etc/hosts*. Por lo cual, el archivo */etc/hosts* es donde se colocan las direcciones locales o también sirve para acceder a *hosts* sin tener que consultar al DNS. La consulta es mucho más rápida, pero tiene la desventaja de que si el *host* cambia, la dirección será incorrecta. En un sistema correctamente configurado, sólo deberán aparecer el *host* local y una entrada para la interfaz *loopback*.

Ejemplo de */etc/hosts*

```
$> cat /etc/hosts
127.0.0.1 localhost loopback
192.168.1.2 pirulo.remix.com
```

Para el nombre de una máquina pueden utilizarse alias, que significa que esa máquina puede llamarse de diferentes maneras para la misma dirección IP. En referencia a la interfaz *loopback*, éste es un tipo especial de interfaz que le permite realizar al *host* conexiones consigo mismo. Por ejemplo, para verificar que el subsistema de red funciona sin acceder a la red. Por defecto, la dirección IP 127.0.0.1 ha sido asignada específicamente al *loopback*. Por ejemplo, un comando *ssh 127.0.0.1* conectará con la misma máquina. La configuración de la interfaz de bucle local es muy fácil y generalmente la realizan los *script* de inicialización de red.

Ejemplo de configuración de *loopback*:

```
$> ifconfig lo 127.0.0.1
$> route add host 127.0.0.1 lo
```

En la versión 2 de la biblioteca GNU (*glibc2*) existe un reemplazo importante con respecto a la funcionalidad del archivo */etc/host.conf*. Esta mejora incluye la centralización de información de diferentes servicios para la resolución de nombres, lo cual presenta grandes ventajas para el administrador. Toda la información de consulta de nombres y servicios ha sido centralizada en el archivo */etc/nsswitch.conf*, el cual permite al administrador configurar el orden y las bases de datos de modo muy simple. En este archivo cada servicio aparece con un conjunto de opciones donde, por ejemplo, la resolución de nombres de *hosts* es una de ellas. En éste se indica que el orden de consulta de las bases de datos para obtener el IP del *host* o su nombre será primero el servicio de DNS, que utilizará el archivo */etc/resolv.conf* para determinar la IP del *host* DNS, y en caso de que no pueda obtenerlo, utilizará el de las bases de datos local, el archivo */etc/hosts*. También se puede controlar por medio de acciones (entre corchetes) el comportamiento de cada consulta, por ejemplo:

```
hosts: xfn nisplus dns [NOTFOUND = return] files
```

Esto indica que cuando se realice la consulta al DNS, si no existe un registro para esta consulta, retorne al programa que la hizo con un cero. Puede utilizarse el ‘!’ para negar la acción, por ejemplo:

```
hosts dns [!UNAVAIL = return] files
```

9.3. Configuración del encaminamiento

Otro aspecto que hay que configurar es el encaminamiento. Si bien existe el tópicos sobre su dificultad, generalmente se necesitan unos requerimientos de encaminamiento muy simples. En un *host* con múltiples conexiones, el encaminamiento consiste en decidir dónde hay que enviar y qué se debe recibir. Un *host* simple con una sola conexión de red también necesita encaminamiento, ya que todos los *hosts* disponen de un *loopback* y una conexión de red. Como se explicó en las secciones anteriores, existe una tabla llamada tabla de encaminamiento que contiene filas con diversos campos, pero a efectos de simplicidad existen tres campos sumamente importantes los cuales son: la dirección de destino, el interfaz por donde saldrá el mensaje y la dirección IP a la cual debemos enviar el mensaje para que este llegue a su destino (*Next Host o gateway*).

El comando *route* permite modificar la tabla de encaminamiento para realizar las tareas de encaminamiento adecuadas. Cuando llega un mensaje, se mira su dirección destino, se compara con las entradas en la tabla y se envía por la interfaz en la cual la dirección que mejor coincide con el destino del paquete.

Consideremos, por ejemplo la red de la figura 7.10 en donde suponemos que nuestro *host* está en una red clase C con dirección 192.168.110.0 y tiene la dirección IP 192.168.110.2 y el *router* con conexión a Internet es el 192.168.110.1. La configuración será:

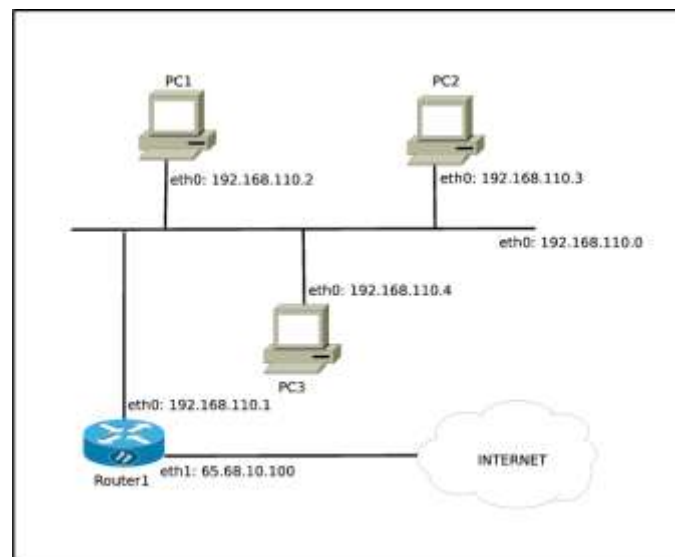


Figura 7.10: Esquema de ejemplo de red

Primero configuramos la interfaz:

```
$> sudo ifconfig eth0 192.168.110.2 netmask 255.255.255.0 up
```

Luego, indicamos que todos los datagramas salientes del *host* 192.168.110.2 y que lleven como destino cualquier *host* dentro de la red 192.168.110.0 deberán ser enviados por el dispositivo eth0 del propio *host*:

```
$> sudo route add -net 192.168.110.0 netmask 255.255.255.0 eth0
```

La opción *-net* en el comando *route* indica que es una ruta de red. Esta configuración permitirá conectarse a todos los *hosts* dentro del segmento de red 192.168.110.0, pero ¿qué pasará si se desea conectar con otro *host* fuera de este segmento? Sería muy difícil tener todas las entradas adecuadas para todas las máquinas a las cuales se quiere conectar. Para simplificar esta tarea, existe la ruta por defecto, que es utilizada como último recurso cuando la dirección destino no coincide con ninguna de las entradas en la tabla:

```
$> sudo route add default gw 192.168.110.1 eth0
```

9.4. Configuración del *inetd*

Uno de los aspectos que debemos configurar en la red es la configuración de los servidores y servicios que permitirán a otro usuario acceder a la máquina local o a sus servicios. Los programas servidores utilizarán los puertos para escuchar las peticiones de los clientes, los cuales se dirigirán a este servicio como IP: port. Los servidores pueden funcionar de dos maneras diferentes: *standalone*: En el cual el servicio escucha en el puerto asignado y siempre se encuentra activo; o a través del *inetd*.

El *inetd* es un servidor que controla y gestiona las conexiones de red de los servicios especificados en el archivo */etc/inetd.conf*, el cual, ante una petición de servicio, pone en marcha el servidor adecuado y le transfiere la comunicación. El objetivo de este servidor es no activar tantos servicios, ya que los servicios de Internet pueden ser numerosos y serían muchos los procesos que estarían en marcha aun sin haber peticiones en curso.

Dos archivos importantes necesitan ser configurados: */etc/services* y */etc/inetd.conf*.

En el archivo */etc/services* se relacionan los servicios con sus correspondientes puertos y protocolos básicos (TCP o UDP en general), ya que la identificación en el nivel de transporte se expresa mediante la dirección IP, el número de puerto y el protocolo utilizado. Es recomendable añadir en este archivo los nuevos servicios locales que se desarrollen.

En el archivo */etc/inetd.conf* se asocia que programas servidores responderán ante una petición a un puerto determinado.

El formato del archivo */etc/services* es:

name port/protocol aliases

La siguiente lista explica cada uno de los campos:

- ***name***: Indica el nombre del servicio.
- ***port/protocol***: Indica el puerto donde se atiende este servicio y el protocolo que utiliza.
- ***aliases***: Especifica un alias del nombre.

Por defecto existen una serie de servicios que ya están pre-configurados. A continuación se muestra un ejemplo de una breve parte del contenido del archivo `/etc/services`:

```
$> head -n 11 /etc/services
tcpmux 1/tcp          # TCP port service multiplexer
echo 7/tcp
echo 7/udp
discard 9/tcp sink null
discard 9/udp sink null
ftp 21/tcp
ssh 22/tcp           # SSH Remote Login Protocol
ssh 22/udp           # SSH Remote Login Protocol
telnet 23/tcp
# 24 - private
smtp 25/tcp mail
```

El símbolo de # indica que lo que existe a continuación es un comentario.

El archivo `/etc/inetd.conf` es la configuración para el servicio maestro de red (*inetd server daemon*). Cada línea contiene siete campos separados por espacios:

```
service socket_type proto flags user server_path server_args
```

La siguiente lista explica cada uno de los campos:

- **service:** Nombre del servicio. Es el servicio descrito en la primera columna del archivo `/etc/services`.
- **socket_type:** Es el tipo de *socket* a utilizar, donde los valores posibles son: *stream* (cuando se utiliza TCP), *dgram* (cuando se utiliza UDP), *raw*, *rdm*, o *seqpacket*.
- **proto:** Es el protocolo válido para esta entrada, este debe coincidir con el de `/etc/services`.
- **flags:** Especifica el modo de servicio que puede ser secuencial o paralelo, es decir, indica la acción que se debe tomar cuando existe una nueva conexión sobre un servicio que se encuentra atendiendo a otra conexión, *wait* (modo secuencial) le dice a *inetd* no poner en marcha un nuevo servidor o *nowait* (modo paralelo) significa que *inetd* debe poner en marcha un nuevo servidor.
- **user:** Será el usuario con el cual se identificará quien ha puesto en marcha el servicio.
- **server_path:** Es la dirección absoluta del ejecutable del servidor. Cuando se trata de un servicio interno de *inetd*, se escribirá *internal* en este campo. Pero si se quiere utilizar el cortafuego *tcpd* se incluirá `/usr/sbin/tcpd` en este campo.
- **server_args:** Son argumentos posibles que serán pasados al servidor.

Un ejemplo de algunas líneas del archivo `/etc/inetd.conf` son:

```
$> cat /etc/inetd.conf
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd
# fsp dgram udp wait root /usr/sbin/tcpd /usr/sbin/in.fspd
shell stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rshd
login stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rlogind
# exec stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rexecd
```

Recordar que # significa comentario, por lo cual, si un servicio tiene # antes del nombre, significa que no se encuentra disponible. Cabe mencionar que al realizar cualquier cambio en este archivo, se deberá reiniciar el servidor `inetd` para que los cambios tengan efectos.

Es importante destacar que existe una versión mejorada del servidor `inetd`, esta versión mejorada es conocida con el nombre de `xinetd` el cual se caracteriza por brindar una mayor garantía de seguridad. Este servidor es el que se suele distribuir con las últimas versiones de GNU/Linux. El archivo de configuración del servidor `xinetd` es el `/etc/xinetd.conf`, en este archivo sólo figuran detalles generales como el número máximo de clientes conectados simultáneamente y la información de seguridad a almacenar, el resto de parámetros aparecerá en el directorio señalado en la última línea.

Un ejemplo de algunas líneas del archivo `/etc/xinetd.conf` son:

```
$> head -n 8 /etc/xinetd.conf
defaults
{
    instances                =        60
    log_type                  =        SYSLOG authpriv
    log_on_success            =        HOST PID
    log_on_failure            =        HOST PID
}
includedir /etc/xinetd.d
```

Por lo tanto, el directorio `/etc/xinetd.d` contendrá un fichero por cada servicio, denominados precisamente con el nombre del servicio. Se expresa con un conjunto de atributos y valores asignados.

La siguiente lista explica cada uno de los atributos esenciales que aparecen:

- **socket type:** El valor será `stream` para el protocolo TCP y `dgram` para el protocolo UDP.
- **wait:** Especifica el modo de servicio, el cual puede ser secuencial o paralelo. Lo único que se debe especificar es `yes` para que el modo sea secuencial o `no` para que el modo sea paralelo. TCP utiliza modo paralelo, por lo que se debe utilizar la opción `no`.
- **user:** Es la cuenta correspondiente al servicio. Suele ser `root`, aunque se suelen utilizar usuarios especiales para algunos servicios o para mejorar la seguridad.
- **server:** Especifica la dirección absoluta del ejecutable del servidor. Los servidores suelen encontrarse en `/usr/sbin`.

- **disable:** Especifica la deshabilitación del servicio. Cuando un servicio se quiere cancelar se debe especificar la opción *yes*.

Por ejemplo, el archivo `/etc/xinetd.d/telnet` tendría el siguiente contenido:

```
$> cat /etc/xinetd.d/telnet
service telnet
{
    flags                =    REUSE
    socket type          =    stream
    wait                 =    no
    user                 =    root
    server               =    /usr/sbin/in.telnetd
    disable              =    no
}
includedir /etc/xinetd.d
```

La función de la última línea es muy importante, ya que permite desactivar el servicio en cuestión. Mientras que en `inetd.conf` un servicio se desactiva comentándolo con el símbolo `#`, aquí se sigue con `disable=yes`, pero ahora en el fichero de servicio no en el `xinetd.conf`.

Al realizar cualquier cambio tanto en el archivo `/etc/xinetd.conf` como en los archivos de servicios, es necesario reiniciar el servidor `xinetd` para que los conseguir que los cambios tengan efectos.

9.5. Configuración adicional: `/etc/protocols` y `/etc/networks`

Existen otros archivos de configuración que en la mayoría de los casos no se utilizan pero que pueden ser interesantes. El `/etc/protocols` es un archivo que relaciona identificadores de protocolos con nombres de protocolos, de esta manera, los programadores pueden especificar los protocolos por sus nombres en los programas.

Ejemplo de `/etc/protocols`:

```
$> head -n 4 /etc/protocols
ip          0    IP          # internet protocol, pseudo protocol number
#hopopt    0    HOPOPT     # IPv6 Hop-by-Hop Option [RFC1883]
icmp       1    ICMP       # internet control message protocol
igmp       2    IGMP       # Internet Group Management
```

El archivo `/etc/networks` tiene una función similar a `/etc/hosts`, pero con respecto a las redes, este presenta una lista de los nombres y direcciones de redes tanto propias como externas. El comando `route` hace uso de este archivo para especificar una red por su nombre en lugar de por su dirección.

Ejemplo de `/etc/networks`:

```
$> cat /etc/networks
loopnet 127.0.0.0
localnet 192.168.0.0
amprnet 44.0.0.0
```

9.6. Algunos aspectos de seguridad a tomar en cuenta

Es importante tener en cuenta los aspectos de seguridad en las conexiones a red, ya que una fuente de ataques importantes se producen a través de la red. En esta sección abarcaremos unas cuantas recomendaciones básicas que deben tenerse en cuenta para minimizar los riesgos inmediatamente antes y después de configurar la red en nuestro sistema.

No debemos activar o dejar activos aquellos servicios que se encuentran en */etc/inetd.conf* que no utilizaremos, podemos insertar un *#* antes del nombre para evitar fuentes de riesgo.

Modificar el archivo */etc/ftpusers* para denegar que ciertos usuarios puedan establecer una conexión vía *ftp* con el sistema.

Modificar el archivo */etc/securetty* para indicar desde qué terminales, un nombre por línea, por ejemplo: *tty1 tty2 tty3 tty4*, se permite la conexión del superusuario *root*. Desde las terminales restantes, *root* no podrá conectarse.

El archivo */etc/hosts.equiv* permite el acceso al sistema sin tener que introducir una contraseña. Se recomienda no utilizar este mecanismo y aconsejar a los usuarios no utilizar el equivalente desde la cuenta de usuario a través de archivo *.rhosts*.

10. Aplicaciones seguras

En esta sección vamos a describir dos aplicaciones de suma importancia para cualquier administrador de sistemas Linux, estas aplicaciones llevan a cabo tareas como administración remota de sistemas (*ssh*) y transferencia de archivos entre sistemas (*scp*). Ambas tareas son de suma importancia en el proceso de administración, pero el problema radica en que necesitamos de la red para llevar a cabo dichas tareas, y como es por todos conocidos la red es una de las principales fuentes de inseguridad de los sistemas. Es por esto que en esta sección llevaremos a cabo dichas tareas mediante aplicaciones que integran servicios de seguridad.

10.1. Secure Shell (*ssh*)

ssh es una aplicación diseñada para sustituir determinadas herramientas de acceso remoto usadas tradicionalmente en los sistemas Linux como son: *telnet*, *rsh* (**R**emote **S**hell), *rlogin* (**R**emote **L**OGIN) o *rcp* (**R**emote **C**oPy). Dicha sustitución es producto de que estas herramientas tradicionales no integran servicios de seguridad entre las comunicaciones, algo que sí se consigue con *ssh*.

El nombre de esta aplicación es la abreviatura de **S**ecure **S**hell, que viene a significar una versión segura del programa **R**emote **S**hell.

La aplicación *ssh* utiliza el protocolo del mismo nombre para llevar a cabo una transmisión segura de los datos. Este protocolo se sitúa directamente por debajo de la capa de transporte (ver figura 7.11) y proporciona servicios análogos a los del protocolo *SSL* (**S**ecure **S**ockets **L**ayer)/*TLS* (**T**ransport **L**ayer **S**ecurity) útiles para conseguir comunicaciones seguras con *http*.

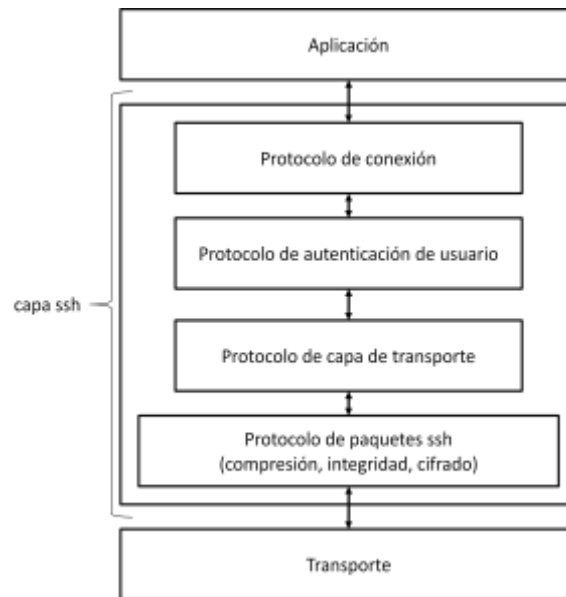


Figura 7.11: Estructura de la capa de transporte *ssh*

ssh sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también es capaz de redirigir el tráfico de *X* para poder ejecutar programas gráficos si se posee un servidor *X* arrancado.

Además de la conexión a otras máquinas, *ssh* nos permite copiar datos de forma segura, tanto ficheros sueltos como simular sesiones *ftp* cifradas. También nos permite pasar los datos de cualquier otra aplicación por un canal seguro utilizando túneles.

El autor de la primera implementación de *ssh* fue Tatu Ylönen de la Universidad Tecnológica de Helsinki, él publicó en el año de 1995 la especificación de la versión 1 del protocolo. Desde entonces se ha trabajado en la especificación de una nueva versión del protocolo, específicamente sobre la versión 2.0. Aunque la funcionalidad que proporciona esta nueva versión es básicamente la misma que la anterior, la nueva versión incorpora muchas mejoras y es sustancialmente distinta de la anterior. La versión antigua y la nueva del protocolo habitualmente son referenciadas como *ssh1* y *ssh2*, respectivamente.

10.1.1. Características del protocolo *ssh*

ssh proporciona los siguientes servicios de seguridad los cuales son equivalentes a los del protocolo *SSL/TLS*.

Confidencialidad. *ssh* sirve para comunicar datos, que habitualmente son la entrada de una aplicación remota y la salida que genera, o bien la información que se transmite por un puerto redirigido, y la confidencialidad de estos datos se garantiza mediante el cifrado.

En *ssh* se aplica un cifrado simétrico a los datos y, por lo tanto, será necesario realizar previamente un intercambio seguro de claves entre cliente y servidor. En *ssh2* se pueden utilizar algoritmos de cifrado distintos en los dos sentidos de la comunicación.

Un servicio adicional que proporciona *ssh* es la confidencialidad de la identidad del usuario. En *ssh* la autenticación del usuario se realiza cuando los paquetes ya se mandan cifrados. Por otro lado, *ssh2* también permite ocultar ciertas características del tráfico como, por ejemplo, la longitud real de los paquetes.

Autenticación de entidad. El protocolo *ssh* proporciona mecanismos para autenticar tanto el ordenador servidor como el usuario que se quiere conectar. La autenticación del servidor suele realizarse conjuntamente con el intercambio de claves. En *ssh2* el método de intercambio de claves se negocia entre el cliente y el servidor, aunque actualmente sólo hay uno definido, basado en el algoritmo de *Diffie-Hellman*.

Para autenticar al usuario existen distintos métodos; dependiendo de cuál se utilice, puede ser necesaria también la autenticación del ordenador cliente, mientras que otros métodos permiten que el usuario debidamente autenticado acceda al servidor desde cualquier ordenador cliente.

Autenticación de mensaje. En *ssh2* la autenticidad de los datos se garantiza añadiendo a cada paquete un código MAC calculado con una clave secreta. También existe la posibilidad de utilizar algoritmos MAC distintos en cada sentido de la comunicación.

Cabe destacar que *ssh* también está diseñado con los siguientes criterios adicionales:

Eficiencia. *ssh* contempla la compresión de los datos intercambiados para reducir la longitud de los paquetes. *ssh2* permite negociar el algoritmo que se utilizará en cada sentido de la comunicación, aunque solamente existe uno definido en la especificación del protocolo. Este algoritmo es compatible con el que utilizan programas como *gzip*.

En *ssh* no está prevista la reutilización de claves de sesiones anteriores: en cada nueva conexión se vuelven a calcular las claves. Esto es así porque *ssh* está pensado para conexiones que tienen una duración más o menos larga, como suelen ser las sesiones de trabajo interactivas con un ordenador remoto. De todas formas, *ssh2* define mecanismos para intentar acortar el proceso de negociación.

Extensibilidad. En *ssh2* también se negocian los algoritmos de cifrado, de autenticación de usuario, de MAC, de compresión y de intercambio de claves. Cada algoritmo se identifica con una cadena de caracteres que representa su nombre. Los nombres pueden corresponder a algoritmos oficialmente registrados, o bien a algoritmos propuestos experimentalmente o definidos localmente.

10.1.2. ¿Por qué usar *ssh*?

ssh proporciona una exhaustiva autenticación y comunicaciones seguras en redes no seguras, es decir, que se usan técnicas de cifrado para que la información vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión.

ssh es compatible con varios algoritmos de cifrado, como pueden ser: *BlowFish*, *Triple DES*, *IDEA*, *RSA*, etc. dicha compatibilidad es algo más que un sencillo adorno de ventanas. Los autores incorporaron esta compatibilidad para crear un protocolo más flexible y ampliable. La arquitectura que implementa *ssh* es tal que al protocolo básico le da igual el algoritmo que se utilice. Por tanto, si posteriormente se descubre que uno o varios de los algoritmos de cifrado compatibles con *ssh* tienen defectos en sus fundamentos o han sido *hackeados*, es posible cambiar rápidamente de uno a otro sin modificar el protocolo clave y las funciones de *ssh*.

ssh no modifica mucho las rutinas. En todos los aspectos, iniciar una sesión de *ssh* es tan sencillo como iniciar una sesión de *telnet*. Tanto la autenticación como el posterior cifrado de sesiones son transparentes.

10.1.3. Ejemplos de uso

Conectarnos remotamente:

```
$> ssh <usuario>@<DireccioIP>
```

O también:

```
$> ssh -l <usuario> <DireccioIP>
```

Ejecutar un único comando sin necesidad de conectarse en el sistema:

```
$> ssh <usuario>@<DireccioIP> <Comando>
```

Ejecutar una aplicación X en el sistema remoto pero utilizando el *display* local:

```
$> ssh -X <usuario>@<DireccioIP> <Aplicación>
```

El parámetro *usuario* debe corresponder con un usuario existente en la máquina remota.

10.2. Secure Copy (*scp*)

scp es una aplicación diseñada para llevar a cabo transferencias seguras de archivos informáticos entre un *host* local y otro *host* remoto, todo esto haciendo uso del protocolo *ssh* visto en la sección anterior.

El término *scp* puede referir a dos conceptos relacionados, el protocolo *scp* o la aplicación *scp*.

10.2.1. El Protocolo *scp*

Es un protocolo simple que deja al servidor y al cliente tener múltiples conversaciones sobre una conexión TCP normal. Esta diseñado para ser sencillo y fácil de implementar.

En el protocolo los datos son cifrados durante su transferencia, para evitar que potenciales *packet sniffers* extraigan información útil de los paquetes de datos. Sin embargo el protocolo por si mismo no provee autenticación y seguridad; sino que espera que el protocolo subyacente, el cual es *ssh*, lo asegure.

El protocolo *scp* implementa la transferencia de archivos únicamente. Para ello se conecta al host usando *ssh* y allí ejecuta un servidor *scp*. Generalmente el programa *scp* del servidor es el mismo que el del cliente.

Para realizar el envío de archivos, el cliente proporciona al servidor los archivos que desea subir y opcionalmente puede incluir otros atributos, como pueden ser permisos, fechas, etc.

Para descargar archivos, el cliente envía una solicitud por los archivos que desea descargar. El proceso de descarga está dirigido por el servidor y es él, el que se encarga de la seguridad de los mismos.

10.2.2. La aplicación *scp*

El cliente *scp* más ampliamente usado es la aplicación *scp* del intérprete de comandos, que es incorporado en la mayoría de las implementaciones de *ssh*. La aplicación *scp* es el análogo seguro del comando *rcp*.

10.2.3. Ejemplos de uso

Copiar un archivo desde el ordenador remoto al ordenador local:

```
$> scp <usuario>@<DireccionIP>:<RutaRemota/ArchivoRemoto> <RutaLocal>
```

Copiar un archivo desde el ordenador local al ordenador remoto:

```
$> scp <ArchivoLocal> <usuario>@<DireccionIP>:<RutaRemota>
```

Copiar una carpeta completa desde el ordenador remoto al ordenador local:

```
$> scp -r <usuario>@<DireccionIP>:<RutaRemota/CarpetaRemota>
```

Copiar una carpeta completa desde el ordenador local al ordenador remoto:

```
$> scp -r <RutaLocal/CarpetaLocal> <usuario>@<DireccionIP>:<RutaRemota>
```

Si la conexión es lenta y lo que transmitimos es fácilmente comprimible:

```
$> scp -C <ArchivoLocal> <usuario>@<DireccionIP>:<RutaRemota>
```

El parámetro *usuario* debe corresponder con un usuario existente en la máquina remota.

TEMA 8: EJECUTAR UN SISTEMA SEGURO

Objetivos

- Comprender las formas básicas de ataques a los que se deben enfrentar los administradores de sistemas.
- Capacitar en las técnicas de prevención básicas para evitar ciertos ataques en los sistemas Linux.
- Conocer y diferenciar las herramientas que sirven al administrador para mantener la seguridad del sistema.

Contenido

1. Una perspectiva sobre la seguridad del sistema
2. Algunos aspectos de seguridad a tomar en cuenta
 - 2.1. Cierre de demonios de red no deseados
 - 2.2. Evitar las amenazas de ingeniería social
 - 2.3. Cumplimiento de las normas de seguridad
3. Las 10 cosas que nunca se deben hacer
4. Configuración del envoltorio TCP
 - 4.1. Utilizar envoltorios TCP con *inetd*
 - 4.2. Utilizar envoltorios TCP con *xinetd*
 - 4.3. */etc/hosts.allow* y */etc/hosts.deny*
5. Cortafuegos: filtrado de paquetes IP
6. Otros elementos útiles para asegurar nuestro sistema
 - 6.1. El servicio *finger* para descubrimiento de usuarios
 - 6.2. Fortaleza de contraseñas con *crack*
 - 6.3. Comprobación proactiva de contraseñas
 - 6.3.1. Comprobador proactivo de contraseñas: *passwd+*
 - 6.3.2. Comprobador proactivo de contraseñas: *anlpasswd*
 - 6.3.3. Comprobador proactivo de contraseñas: *npasswd*
 - 6.4. Mapeo de puertos con *nmap*
 - 6.5. Sistemas de detección de intrusos
 - 6.5.1. Detección de ataques en red con *snort*
 - 6.6. Advertencias ante paquetes con *bugs*
 - 6.7. Auditando vulnerabilidades con *nessus*

Bibliografía

Básica

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.
- Jordi Herrera Joancomartí, Joaquín García Alfaro, Xavier Perramón Tornil, “Aspectos avanzados de seguridad en redes”, UOC Formación de Posgrado, Software libre, 2004.

Complementaria

- Dee-Ann LeBlanc, “Administración de sistemas LINUX La biblia”. Editorial ANAYA MULTIMEDIA, 2001.
- Jack Tackett Jr. y David Gunter, “Linux Tercera Edición, Edición Especial”, Editorial Prentice Hall, 1998.
- Ataque de denegación de servicio.
http://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

- Script Kiddie.
http://es.wikipedia.org/wiki/Script_kiddie
- Nmap.
<http://es.wikipedia.org/wiki/Nmap>
<http://nmap.org/>
- SNORT.
<http://es.wikipedia.org/wiki/SNORT>
<http://www.snort.org/>
- Nessus.
<http://es.wikipedia.org/wiki/Nessus>
<http://www.nessus.org/nessus/>

En este tema vamos a analizar la seguridad básica que como buenos administradores debemos añadir a nuestro sistema Linux. La seguridad siempre es un tema muy importante, especialmente con el uso creciente de sistemas que se encuentran conectados permanentemente a redes y que por lo tanto son vulnerables a los ataques remotos.

Gran parte de la seguridad del sistema depende de utilizar el sentido común en relación con la disponibilidad de una serie de herramientas informáticas. Muchas de las mejores técnicas son las más sencillas, aunque en muchas circunstancias son también las más ignoradas.

1. Una perspectiva sobre la seguridad del sistema

En ocasiones es difícil mantener una perspectiva equilibrada sobre la seguridad del sistema. Los medios de comunicación o incluso sitios en Internet suelen presentar historias sensacionalistas sobre las brechas de seguridad, especialmente cuando se ven implicadas empresas o instituciones que son mundialmente conocidas. Por otro lado, administrar la seguridad puede ser una tarea técnicamente desafiante y a la que tenemos que dedicar mucho tiempo. Muchos usuarios de Internet creen que su sistema no contiene datos valiosos, por lo que la seguridad no les supone un problema, lo cual es una mala suposición. Otros dedican mucho esfuerzo para proteger sus sistemas frente a un uso no autorizado. Independientemente del espectro al que pertenezcamos, debemos tener en cuenta que siempre hay un riesgo de convertirnos en el objetivo de un ataque de seguridad. Existen muchas razones por las que alguien pudiera estar interesado en romper la seguridad de nuestro sistema, el valor de los datos que contienen los sistemas es sólo una de ellas.

Los usuarios son libres de tener su propia opinión sobre la cantidad de esfuerzo que dedican a la implementación de medidas de seguridad sobre sus sistemas. Pero los administradores no pueden ni deben darse esos lujos, pues deben ser siempre precavidos, cautelosos y oportunos.

Anteriormente, la seguridad tradicional del sistema se centraba en los sistemas a los que se podía acceder a través de un terminal conectada por cable o a través de una consola del sistema. En este ámbito, los riesgos más importantes provenían normalmente de la propia organización propietaria del sistema y la mejor forma de defensa era la seguridad física, en la que consolas, terminales, y servidores del sistema se colocaban en estancias cerradas con llave. Incluso cuando los sistemas de equipos empezaron a conectarse en red, el acceso seguía siendo muy limitado. Las redes que se utilizaban normalmente eran muy caras como para acceder a ellas o eran redes cerradas que no permitían conexiones a servidores de cualquier parte. Hoy en día, la popularidad de Internet ha hecho surgir una nueva generación de problemas de seguridad basados en la red. Un equipo conectado a Internet está abierto a un abuso potencial por parte de decenas de millones de equipos de todo el mundo. Con la accesibilidad mejorada y tan amplia llega un incremento en el número de individuos antisociales que intentan causar un perjuicio. En Internet, existen diversas formas de comportamiento antisocial de interés para los administradores de sistemas. Las más comunes son las siguientes:

- **Denegación de servicio (DoS, Denial of Service):** Este tipo de ataque puede ser dirigido a un sistema de ordenadores o a una determinada red, causa que un servicio o recurso sea degradado o interrumpido a los usuarios legítimos. Normalmente provoca la degradación o pérdida, en el peor de los casos, de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la víctima.

Un ataque de *DoS* implica normalmente generar una gran cantidad inusual de solicitudes de servicios proporcionados por un sistema. Esta ráfaga de actividad puede hacer que el sistema agote su memoria, potencia de procesamiento o ancho de banda de la red. Otra forma es proporcionar servicio con una entrada poco habitual para explotar

un fallo y producir un volcado de memoria. Como resultado, se rechazan las solicitudes posteriores al sistema o se degrada el rendimiento del mismo hasta un punto inusual. Para que funcione este tipo de ataque, un atacante tiene que explotar un servicio mal diseñado o generar muchas solicitudes que excedan la capacidad del servicio.

Una forma más insidiosa de ataque *DoS* es la denegación de servicio distribuido (*DDoS*, *Distributed Denial of Service*). En esta forma de ataque, se utilizan muchos equipos para generar solicitudes a un servicio. Así se incrementa el daño del ataque *DoS* de dos formas: la primera sobrecargando el objetivo con un gran volumen de tráfico y la segunda ocultando al autor del ataque tras miles de participantes involuntarios. Al utilizar una gran cantidad de equipos para lanzar un ataque, los ataques *DDoS* son particularmente difíciles de controlar y solucionar cuando se han producido. Incluso los usuarios a los que no les preocupa la seguridad de sus propios datos, deben protegerse frente a este tipo de ataque para minimizar el riesgo de convertirse en un cómplice involuntario de un ataque *DDoS* frente a un tercero.

- **Intrusión:** Este tipo de ataque accede al sistema adivinando las contraseñas o comprometiendo algún servicio. Cuando el intruso consigue acceder al sistema, puede robar datos, utilizar el sistema de destino para lanzar ataques sobre algún otro sistema, etc.

Este tipo de ataque es conocido a veces como *cracking*, y es la que la mayoría de usuarios asocian con la seguridad. Los términos *cracking* y *hacking* normalmente se confunden en su uso popular. El primero implica una conducta no ética o ilegal, como comprometer la seguridad del sistema, y el segundo es una palabra genérica que significa programar, ajustar o tener un gran interés en algo. Los medios de comunicación normalmente utilizan el término *hacking* para referirse a *cracking*. Las empresas e instituciones normalmente guardan datos confidenciales en sistemas de equipos accesibles a través de la red. Un ejemplo común de preocupación para el usuario medio de Internet es el almacenamiento de los detalles de tarjetas de crédito en los sitios web. Siempre que haya dinero implicado, existe un incentivo para que las personas deshonestas obtengan acceso y roben o utilicen este tipo de datos confidenciales para su propio provecho.

- **Escucha clandestina de paquetes:** Este tipo de ataque implica interceptar los datos de otros usuarios y obtener las contraseñas u otra información confidencial. Algunas veces, esta forma de ataque también implica la modificación de los datos interceptados que viajan por la red. La escucha clandestina de paquetes normalmente implica espiar y analizar a escondidas las conexiones de red, pero también se puede comprometer un sistema para que intercepte las llamadas de bibliotecas o del sistema, que transportan información confidencial, por ejemplo: las contraseñas.
- **Virus, gusanos y troyanos:** Estos ataques se ven apoyados por los usuarios del sistema ya que estos ataques suelen suceder cuando los usuarios del sistema ejecutan programas suministrados por el atacante. Los programas se pueden recibir a través de un mensaje de correo electrónico, de un sitio web o incluso de otro programa, aparentemente inofensivo, recuperado de alguna parte de Internet e instalado localmente.

Algunas veces, los métodos que se utilizan para obtener un acceso no autorizado o para interrumpir un servicio son muy ingeniosos, por no decir poco éticos. Normalmente el diseño de un mecanismo de intrusión requiere un buen conocimiento del sistema de destino para descubrir un fallo explotable. Normalmente, tras el descubrimiento de un mecanismo de intrusión, éste se empaqueta en forma de *rootkit*, los cuales son un conjunto de programas o secuencias de comandos que cualquiera con conocimientos básicos puede utilizar para explotar una determinada brecha de seguridad. La gran mayoría de ataques de intrusión los lanzan los

“*chicos de la secuencia de comandos*” (*script kiddies*), que utilizan estos conjuntos de intrusión empaquetados sin ningún conocimiento real del sistema al que están atacando. Lo bueno es que normalmente es muy fácil para un administrador de sistemas ejecutar una protección frente a este tipo de ataques muy conocidos.

2. Algunos aspectos de seguridad a tomar en cuenta

Podemos hacer muchas cosas para asegurar nuestro sistema Linux de los riesgos de seguridad más básicos. Evidentemente, dependiendo de la configuración del sistema, de la forma en la que este se va a utilizar, de la forma en que la que se va a acceder, de los tipos de usuarios dentro del mismo, etc. puede que tengamos que seguir algunos pasos adicionales, basados en el nivel de seguridad que queremos aplicar sobre el sistema. En esta sección vamos a analizar brevemente algunos aspectos básicos que debemos tener en cuenta para asegurar nuestro sistema Linux.

2.1. Cierre de demonios de red no deseados

El primer paso para asegurar un sistema Linux es cerrar o deshabilitar todos los demonios y servicios de red que no se vayan a necesitar. Básicamente, cualquier puerto de red (externo) al que está escuchando el sistema por conexiones, es un riesgo, ya que podría haber una explotación de seguridad frente al demonio que sirve dicho puerto. La forma más rápida de descubrir cuáles son los puertos abiertos es utilizar: `sudo netstat -an`.

Por ejemplo:

```
$> sudo netstat -an
Conexiones activas de Internet (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp    0      0 0.0.0.0:139      0.0.0.0:*      LISTEN
tcp    0      0 0.0.0.0:6000     0.0.0.0:*      LISTEN
tcp    0      0 0.0.0.0:22      0.0.0.0:*      LISTEN
```

En este ejemplo podemos ver que este sistema está escuchando conexiones en los puertos 139, 6000 y 22. Si examinamos el archivo `/etc/services` normalmente podemos ver qué demonios están asociados con dichos puertos. En este caso son el servicio de sesión de *NETBIOS* (puerto 139), el sistema *X Window* (puerto 6000) y el demonio *ssh* (puerto 22).

Si en nuestro sistema observamos que existen muchos otros puertos abiertos, para servicios como *telnetd*, *sendmail*, etc., debemos valorar si realmente necesitamos que se estén ejecutando dichos demonios y que estos sean accesibles desde otros ordenadores. De vez en cuando se anuncian explotaciones de seguridad para diversos demonios y, a no ser que seamos muy buenos manteniendo un registro de estas actualizaciones de seguridad, nuestro sistema podría ser vulnerable a un ataque.

Normalmente, para cerrar un servicio, hay que desinstalar el paquete correspondiente al servicio. Si deseamos mantener el cliente pero el cliente y el demonio (servidor) están empaquetados juntos, algo que suele ser actualmente extraño en las distribuciones de Linux, tendremos que editar los archivos de configuración apropiados para nuestra distribución y reiniciar el sistema (para ello debemos asegurarnos de que el demonio es bueno y está cerrado). Por ejemplo, en los sistemas Debian, muchos demonios se inician a través de secuencias de comandos ubicados en el directorio `/etc/init.d`; si renombramos o eliminamos estas secuencias de comandos, podremos evitar que se inicien los demonios correspondientes. Otros demonios se inician a través de *inetd* o *xinetd* en respuesta a conexiones de red entrantes; si modificamos la

configuración de estos sistemas, podremos limitar el conjunto de demonios que se ejecutan en el sistema.

Si es absolutamente necesario que se ejecute un servicio en nuestro sistema, como por ejemplo el servidor X, debemos buscar la forma de evitar que las conexiones a dichos servicios se realicen a través de ordenadores no deseados. Por ejemplo, puede ser más seguro permitir las conexiones *ssh* sólo de determinados ordenadores de confianza, como de máquinas que estén ubicadas en nuestra red local. En el caso del servidor X y el servidor de fuentes X que se ejecutan en muchas máquinas de sobremesa Linux, normalmente no existe ninguna razón para permitir las conexiones a dichos demonios desde ningún lugar que no sea el propio servidor local. El filtrado de conexiones para dichos demonios se puede llevar a cabo a través de envoltorios TCP o de filtrado IP.

2.2. Evitar las amenazas de ingeniería social

Con todas las funciones de seguridad con las que cuenta el sistema Linux, puede decirse que son sus usuarios quienes más comprometen la seguridad del sistema. Después de todo, ellos disponen de una cuenta válida la cual les brinda su acceso al sistema.

Pero, ¿qué tiene que ver esto con la ingeniería social? Y de hecho, ¿qué es la ingeniería social? La ingeniería social se refiere al deseo de convencer a otras personas para que hagan lo que uno quiera, influyendo sobre sus creencias o comportamientos o mediante falsedades y mentiras. Las personas, por norma general, desean ser de utilidad. Y, si se les da la oportunidad, normalmente tratan de ayudar lo más posible. Los *crackers* con buenos dotes de ingeniería social se aprovechan de la amabilidad de las personas para conseguir la información que ellos necesitan.

Examinemos un ejemplo. Supongamos que dentro de nuestro sistema existe un usuario informático llamado Juan, que es un usuario medio, no un experto. Un día el usuario Juan recibe una llamada en la oficina, similar a la que se detalla a continuación:

Usuario Juan: Diga.
Persona que llama: Buenos días, señor Juan. Soy Carlos García, de soporte técnico. Lo llamaba para indicarle que debido a algunas limitaciones de espacio de disco vamos a transferir algunos directorios locales de usuario a otro disco a las 17:30 horas. Su cuenta también será transferida, por lo que no se encontrará disponible durante algún tiempo.
Usuario Juan: Bueno, de acuerdo. De todas formas a esa hora ya estaré en mi casa.
Persona que llama: Muy bien. No olvide desconectarse del sistema antes de marcharse. Sólo necesito comprobar un par de cosas. ¿Cuál es su identificador de conexión?, ¿juan?
Usuario Juan: Efectivamente, es juan. Pero no se perderá ninguno de mis archivos durante la transferencia, ¿verdad?
Persona que llama: No, señor. Pero comprobaré su cuenta de todas formas para estar seguro. Dígame la contraseña de la cuenta para que pueda comprobar sus archivos.
Usuario Juan: Mi contraseña es martes.
Persona que llama: Muy bien, señor Juan. Gracias por su ayuda. Me aseguraré de comprobar su cuenta y verificar que todos los archivos se encuentren en su sitio.
Usuario Juan: Gracias y adiós.

¿Qué ha sucedido? Sencillamente, alguien ha llamado a uno de los usuarios de nuestro sistema por teléfono y ha obtenido un nombre válido de usuario y la contraseña durante la

conversación. Y probablemente el señor Juan llamará al día siguiente al centro de soporte técnico y le dirán que no hay ninguna persona llamada Carlos García que trabaje allí.

¿Cómo pueden evitarse este tipo de circunstancias? Como buenos administradores del sistema debemos informar a nuestros usuarios del riesgo que corren al liberar información confidencial a personas que no conocen. Esto nos lleva a cabo implementaciones de políticas de seguridad a nivel de usuarios como por ejemplo: Indicarles que nunca deben facilitar su contraseña por teléfono a nadie, o mejor dicho que nunca deben facilitar su contraseña a personas que no demuestren ser aquellas quien dicen ser. Otra política puede ser enseñarles a los usuarios a nunca dejar sus contraseñas en el correo electrónico o de voz. Los *crackers* utilizan la ingeniería social para convencer a los usuarios de que les den lo que les piden y ni siquiera tienen que introducirse en nuestro sistema para conseguirlo, algo que juega a su favor pues así no dejan vestigios de sus intentos fallidos para conseguir acceso al sistema.

2.3. Cumplimiento de las normas de seguridad

Es lógico que las empresas de la industria de defensa cuenten con complejos sistemas de seguridad para protegerse. Igualmente, las empresas con productos muy delicados en el proceso de diseño son consientes de la necesidad de protegerse adecuadamente de posibles filtraciones. Pero, por ejemplo, a los empleados de una pequeña empresa distribuidora de piezas de fontanería les puede resultar más difícil entender por qué todo el mundo está tan preocupado por la seguridad. De hecho no se preocuparán por ella hasta que no sepan quién se ha podido llevar un archivo que incluía una propuesta clave para la empresa.

A los empleados debería sensibilizárseles sobre la importancia de los datos que contiene(n) la(s) máquina(s) de la empresa con la cual trabajan. De hecho, los datos almacenados allí suponen una parte importante de la inversión de la empresa. La pérdida de los mismos podría significar una pequeña desorganización o el caos total. Por todo esto, es fundamental hacer ver a los empleados la necesidad de tomar las debidas precauciones de seguridad y que su descuido o desinterés puede ser causa de despido.

Para un administrador, la seguridad de una red es una cuestión vital. Pero, ¿cómo puede estar seguro de que los archivos y directorios se encuentran adecuadamente protegidos? Afortunadamente existen muchas herramientas diseñadas para ayudarnos con esta labor, como, por ejemplo, *umask*, *cron*, *chmod* y el propio Linux.

Los permisos suelen ser una fuente de preocupaciones para casi todos los administradores. Los administradores más novatos normalmente hacen más difícil el acceso a permisos y luego atienden las llamadas de aquellos usuarios que dicen no poder acceder a un archivo que necesitan o dicen no poder ejecutar un programa en el sistema. Después de algún tiempo, estos administradores facilitan los permisos hasta que todo el mundo puede hacer cualquier cosa. Encontrar el justo equilibrio entre las medidas que deben tomarse para proteger el sistema y las herramientas que deben proporcionarse para que los usuarios autorizados puedan llevar a cabo sus trabajos, no es tarea fácil y puede llegar a ser bastante frustrante.

3. Las 10 cosas que nunca se deben hacer

Se ha señalado que la seguridad es principalmente una cuestión de sentido común, por tanto, ¿cuál es este sentido común que como administradores debemos tener? A lo largo de esta sección vamos a presentar un resumen de los errores de seguridad más comunes que como buenos administradores de sistemas Linux no debemos cometer. Evitarlos de forma consistente es una tarea más difícil de lo que parece.

No utilizar contraseñas simples o que se puedan adivinar con facilidad

Una de las maneras básicas por la que muchos administradores y usuarios debilitan la seguridad del sistema es debido a la elección de contraseñas pobres o simples. Las contraseñas constituyen nuestra primera línea de defensa contra el intruso en el sistema. Incluso el menos entendido de los posibles intrusos sabe que si lo intenta lo suficiente tiene una posibilidad de descubrir la contraseña de alguien. Existen programas tanto a disposición nuestra como de los atacantes que están escritos con el objetivo de descifrar contraseñas débiles o simples, por ejemplo: *Crack*.

Algunas directrices o consideraciones a tener en cuenta a la hora de elegir una buena contraseña pueden ser:

- **No utilizar palabras de diccionario:** Estas se comprueban fácilmente con archivos de texto maestros utilizando una gran variedad de programas de crack de contraseñas. Las mejores contraseñas son las formadas por cadenas sin sentido. Una buena práctica es utilizar contraseñas basadas en una simple regla y en una frase que sea fácil de recordar. Por ejemplo, podemos elegir una regla que utilice la última letra de cada palabra de la frase "*María tenía una pequeña ovejita y era tan blanca como la nieve*"; de ahí, la contraseña se convertiría en *aaaaayanaoae*, algo que evidentemente no es fácil de adivinar por posibles atacantes y es fácilmente recordable por nosotros.
- **Utilizar combinaciones de mayúsculas y minúsculas, números y símbolos permitidos:** Muchos usuarios utilizan automáticamente letras minúsculas para escribir con comodidad, o simplemente escriben la primera letra con mayúscula. Esto es demasiado predecible. Cuanto más se mezclen las cosas en la contraseña, más segura resultará.
- **No utilizar información personal:** Alguien puede piratear un sistema porque previamente hizo investigaciones y probó con toda la información personal de un determinado usuario de cuenta como variantes de la contraseña. Algunos ejemplos pueden ser: no utilizar nunca una contraseña que sea igual, o bastante parecida al: ID de usuario, nombre, fecha de nacimiento, nombre de la empresa, nombre de mascotas personales, etc. En resumen nada de información personal.
- **No escribir la contraseña en un papel y luego esconderla en el cajón del escritorio:** Muchos usuarios tienen la mala costumbre de no poner en práctica esta regla, cabe mencionar que por muy segura que sea la contraseña, si no cumplimos con esta regla de nada nos servirá generar una contraseña extremadamente segura, pues le dejaremos en bandeja de oro el acceso a las personas que estén alrededor de nuestro espacio de trabajo.

No utilizar la cuenta de root a no ser que sea estrictamente necesario

Una de las razones por las que muchos sistemas operativos de sobremesa comunes, por ejemplo Windows, son tan vulnerables a los ataques de virus de mensajes de correo electrónico es la falta de un sistema de privilegios amplio, o la comodidad del usuario de ejecutar aplicaciones con privilegios de administrador. Debemos tener cuidado cuando algunas aplicaciones mal intencionadas requieran que se ejecuten con derechos de administrador. En este tipo de sistemas, los usuarios son capaces de acceder a cualquier archivo, ejecutar el programa que deseen o volver a configurar el sistema. Por ello, es fácil obligar a un usuario a ejecutar un programa que pueda provocar un daño real en el sistema. En contraste, el modelo de seguridad de los sistemas Linux limita un amplio rango de tareas con privilegios, como la instalación de un nuevo software o la modificación de archivos de configuración, al usuario *root*. *¡No debemos caer en la mala práctica de usar la cuenta root para todo!* al hacerlo estamos desaprovechando una de las defensas más poderosas contra los ataques de virus y

troyanos, sin mencionar los comandos `rm -rf *` accidentales. Debemos utilizar siempre una cuenta de usuario normal y utilizar los comandos `su` o `sudo` para obtener un acceso temporal como `root` cuando necesitemos realizar tareas con privilegios. Existe una ventaja adicional en este limitado uso de la cuenta `root`: los registros. Los comandos `su` y `sudo` escriben mensajes en el archivo de registro del sistema, específicamente en `/var/log/auth.log`, cuando se les llama, mencionando el ID del usuario que ejecuta el comando así como la fecha y la hora en que se ha llamado al comando, algo muy útil para supervisar cuando se han utilizado los privilegios de `root` y quién los ha utilizado.

No compartir las contraseñas

No decir a nadie cuales son las contraseñas de acceso al sistema, nunca. Si deseamos que alguien acceda temporalmente al sistema, lo mejor es crear una cuenta para que la utilice dicha persona. Si realmente necesitamos confiarle la cuenta de `root` a otra persona, lo mejor es utilizar el comando `sudo`, el cual nos permite proporcionar a los usuarios un acceso `root` a ciertos comandos seleccionados por nosotros sin necesidad de revelar la contraseña de `root`.

No creer ciegamente en los binarios proporcionados

Aunque es un método muy cómodo para recuperar e instalar copias binarias de programas en el sistema, siempre debemos cuestionarnos la confianza en el binario antes de ejecutarlo. Si estamos instalando paquetes recuperados directamente de los sitios oficiales de la distribución o de sitios de desarrollo importantes, es muy probable que el software sea seguro. Si los estamos recuperando desde un sitio espejo no oficial, tendremos que considerar cuánto confiamos en los administradores de dichos sitios. Es posible que exista alguien que este distribuyendo una forma modificada del software con puertas traseras que permitan a alguien obtener acceso a nuestro sistema. Aunque pueda parecer un punto de vista algo paranoico, cada día se suman más a la erradicación de distribuciones de formas modificadas de software, organizaciones de distribuciones Linux. Por ejemplo, la organización Debian está desarrollando un medio para analizar el paquete de software para confirmar que no ha sido modificado. Otras distribuciones están adoptando técnicas similares para proteger la integridad de su propio software empaquetado.

Si deseamos instalar y ejecutar un programa proporcionado en forma binaria, podemos seguir alguna técnica que nos ayude a minimizar el riesgo. Por ejemplo, en primer lugar, ejecutar siempre los programas que no sean de confianza como un usuario distinto de `root`, a no ser que el programa requiera específicamente privilegios de `root` para funcionar, algo que podremos solucionar si hacemos uso de máquinas destinadas para pruebas utilizando por ejemplo entornos de vitalización como *VMware* o *Virtual Box*. De esta forma podremos evitar cualquier daño potencial que pueda producir el programa. Si deseamos saber lo que hace el programa antes de ejecutarlo, podemos ejecutar el comando `strings` sobre los binarios. Este comando mostrará todas las cadenas integradas que aparecen en el código. Debemos buscar cualquier referencia a archivos o directorios importantes, como `/etc/passwd` o `/bin/login`. Si observamos una referencia a un archivo importante, debemos preguntarnos si tiene que ver con el propio programa. En caso contrario, debemos tener mucho cuidado. También deberíamos considerar primero la ejecución del programa y observar su comportamiento haciendo uso de comandos como `strace` o `ltrace`, que muestran las llamadas de bibliotecas y del sistema que está efectuando el programa. Debemos buscar referencias inusuales al sistema de archivos o una actividad de red poco habitual.

No ignorar los archivos de registro

Los archivos de registro del sistema son nuestros amigos y pueden indicarnos muchas cosas sobre lo que ha sucedido o está sucediendo en el sistema. Podemos encontrar información sobre cuándo se han realizado las conexiones de red para el sistema, quién ha estado intentando

utilizar la cuenta *root* y ha fallado en sus intentos, etc. Debemos revisar periódicamente los archivos de registro y aprender a distinguir lo normal de lo poco habitual. En caso de observar algo inusual, lo mejor es investigar.

No dejar de actualizar el sistema durante mucho tiempo

Es importante mantener actualizado el software en el sistema. Al mantener actualizado el software del sistema nos aseguramos de haber aplicado todas las soluciones a fallos de seguridad. La mayoría de distribuciones de Linux proporcionan un conjunto de paquetes que sólo solucionan fallos de seguridad, por lo que no tendremos que preocuparnos de temas como el archivo de configuración y realizarle cambios para mantener seguro el sistema. Una buena práctica es seguirle las pistas a las actualizaciones.

No debemos olvidarnos de la seguridad física

La mayoría de brechas de seguridad son aprovechadas por personas pertenecientes a la organización que ejecuta el sistema objetivo. La configuración de seguridad de software más amplia en el mundo significa que nadie puede entrar en el sistema y ejecutar un disquete que contenga un código de explotación. Si el sistema utiliza *BIOS* o *PROM* lo más recomendable es configurar los dispositivos de arranque, debemos establecerlos para que el disquete, las unidades de CD-ROM y cualquier otra unidad extraíble se inicien tras el disco duro. Las *BIOS* modernas proporcionan apoyo para la protección con contraseña de la configuración que contienen, así que debemos utilizarla. Estas *BIOS* también proporcionan la asignación de contraseñas a los discos duros, algo que nos permite asegurar que el contenido del mismo va a poder ser únicamente leído por el conocedor de dicha contraseña. Si podemos cerrar el gabinete o *case* de la máquina que contiene el sistema con candado, debemos hacerlo. La idea es mantener físicamente segura la máquina.

No descuidar los permisos de los archivos

Otra manera de facilitarles las cosas a los intrusos es el uso descuidado de los permisos y otras cuestiones de este tipo del sistema de archivos. Debemos recordar que los permisos débiles o no madurados son uno de los modos más rápidos por los que los usuarios entra en zonas del sistema de archivos en las que no deberían entrar.

No olvidar la existencia de sitios web de seguridad

Todos los administradores de sistemas deberían visitar regularmente ciertos sitios web de seguridad, la verdad es que existe mucha información, pero si nos tomamos en serio la seguridad del sistema, que es lo que deberíamos hacer, es necesario que estos sitios sean visitados. Primero, debemos tomarnos el tiempo para recorrer estos sitios con seriedad y después, tranquilamente, debemos hacer una lista de marcadores para las secciones concretas que deseamos comprobar regularmente. Este tipo de sistemas funciona bien hasta que conseguimos organizar los sitios de interés.

Estos son algunos de los sitios web de seguridad general:

- **CERT:** Es un grupo veterano de seguridad de Internet, fundado en 1988 por *DARPA* (*Defense Advanced Research Projects Agency*). Su función inicial era la respuesta de emergencia de seguridad de Internet. Hoy, aplica esa riqueza de conocimientos a investigar lo último en cuanto a necesidades de seguridad en Internet y para ayudar a aumentar los rangos de especialistas de seguridad de Internet y equipos de respuesta de emergencia.

Dirección: <http://www.cert.org/advisories/>

- **CIAC:** *CIAC (Computer Incident Advisory Capability)* es sólo un año más joven que *CERT*, establecido en 1989 por el departamento de energía de los Estados Unidos para ocuparse de cuestiones y educación de seguridad de Internet. Hoy, todavía trabaja con el *DOE (Department Of Energy)*, pero proporciona educación y una vigilancia de tecnología a la comunidad de Internet.
Dirección: <http://www.ciac.org/ciac/index.html>
- **rootshell:** Un sitio de documentación y noticias de seguridad. Normalmente, *rootshell* cuenta con una lista de elementos que incluye sitios que fueron forzados e información a cerca de cómo ocurrió de modo que podamos asegurar mejor nuestros propios sistemas. Existe también una colección de artículos que se encuentran relacionados con diversas cuestiones de seguridad.
Dirección: http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/root_shell/default.htm
- **SANS:** *SANS (System Administration, Networking and Security)* es una organización de educación y difusión de noticias de seguridad. Produce una serie de resúmenes online, libros y papeles físicos, y conferencias de educación de seguridad.
Dirección: <http://www.sans.org/>
- **CVE:** *CVE (Common Vulnerabilities and Exposures)* son siglas que se empiezan a ver cada vez más junto a las últimas vulnerabilidades o *exploits* encontrados, independientemente del tipo de problema o del sistema operativo al que hagan referencia. Básicamente consiste en definir una forma común y estándar de llamar a las vulnerabilidades o *exploits*. *CVE* no debe ser considerada como una base de datos de vulnerabilidades, sino más bien como una especie de listado o diccionario de vulnerabilidades con referencias a los anuncios o avisos oficiales desarrollados por una gran multitud de empresas o entidades relacionadas con la seguridad informática, Tales como: CISCO, COMPAQ, DEBIAN, CERT, FreeBSD, IBM, MANDRAKE, NetBSD, OPENBSD, REDHAT, etc.
Dirección: <http://cve.mitre.org/>
- **hispasec:** Nos brinda la vulnerabilidad más destacada del día, esto nos permitirá estar al tanto en cuanto a vulnerabilidades se refiere y por ende prepararnos para búsqueda de parches o actualizaciones.
Dirección: <http://www.hispasec.com/>

Nunca llevar a cabo una administración remota sin un *shell* seguro

Tenemos a disposición una variedad de *shells* seguros en Linux. Cada uno tiene sus ventajas y sus desventajas. Sin embargo, uno se mantiene firme en la esfera de la seguridad: *ssh*, el *shell* seguro.

Cabe destacar que si hacemos uso de manera regular de *telnet* o *rlogin* para llevar a cabo una administración remota, estaremos entregando nuestro sistema a los posibles atacantes, es por eso que si no deseamos ser considerados unos pésimos administradores de sistemas Linux, jamás, pero jamás, debemos hacer uso de *telnet* o *rlogin*. Pues para llevar a cabo una administración remota deberíamos hacer efectivo el uso de *ssh*, siempre. La filosofía de hacer uso de *ssh* es que toda comunicación entre dos máquinas siempre es codificada, algo que con *telnet* o *rlogin* no sucede, de esta manera evitamos que existan personas que puedan husmear nuestras comunicaciones con el fin de observar y entender lo que estamos haciendo.

4. Configuración del envoltorio TCP

Hemos explicado anteriormente que la conexión de nuestro sistema a una red aumenta considerablemente el riesgo de ataques. Tras presentar las consideraciones de sentido común, es el momento de examinar con más detalle la seguridad básica de una red. En esta sección vamos a analizar un método simple, aunque efectivo, de reducir el riesgo de accesos no deseados a la red utilizando una herramienta denominada envoltorio TCP. Este mecanismo *envuelve* un mecanismo existente, como el servidor de correo, supervisando sus conexiones de red y rechazando las conexiones de sitios no autorizados. Éste es un método muy sencillo de añadir un control de acceso a los servicios que originalmente no estaban diseñados para ello, y normalmente se utilizan junto con los demonios *inetd* o *xinetd*.

Los envoltorios TCP son equivalentes a los guardas de seguridad que se encuentran protegiendo la entrada de las fiestas o locales de diversión importantes. Al llegar un individuo, se encontrará con dicho guarda, que le pedirá su nombre y dirección. El guarda consultará la lista de invitados y, solamente si dicho individuo se encuentra en ella, el guarda se hará a un lado para permitirle que entre a la fiesta.

Cuando se efectúa una conexión de red a un servicio protegido por envoltorios TCP, con lo primero que nos encontramos es con el envoltorio. Éste comprueba el origen de la conexión de red utilizando el nombre de *host* o la dirección de origen y consulta una lista que describe quién puede acceder. Si el origen coincide con una entrada en la lista, el envoltorio se aparta del camino y permite que la conexión de red acceda al programa de demonio real.

Existen dos formas de utilizar envoltorios TCP, dependiendo de la distribución de Linux y de la configuración. Si estamos utilizando el demonio *inetd* para administrar servicios, podemos comprobar si existe el archivo */etc/inetd.conf*, los envoltorios TCP se implantan utilizando un demonio especial denominado *tcpd*. Si estamos utilizando en su lugar el demonio *xinetd*, podemos comprobar si existe el directorio */etc/xinetd.d*, *xinetd* normalmente está configurado para utilizar directamente envoltorios TCP. En las siguientes secciones examinaremos cada caso.

4.1. Utilizar envoltorios TCP con *inetd*

Si nuestro sistema utiliza el demonio *inetd* para abrir servicios de red, puede que tengamos que editar el archivo */etc/inetd.conf* para poder utilizar envoltorios TCP. Vamos a utilizar el demonio de *finger*, *in.fingerd*, como ejemplo. La idea básica es que en lugar de ejecutar el demonio *in.fingerd*, *inetd* labore en su lugar con el demonio *tcpd*. Éste ejecuta la operación del envoltorio TCP y después ejecuta en su lugar *in.fingerd*, si se acepta la conexión.

La configuración de envoltorios TCP requiere un cambio muy sencillo en el archivo */etc/inetd.conf*. Para el demonio *finger*, podemos tener una entrada como la siguiente en este archivo:

```
$> cat /etc/inetd.conf
#/etc/in.fingerd finger daemon
finger stream tcp nowait root /usr/sbin/in.fingerd in.fingerd
```

Para proteger al demonio *finger* con *tcpd*, simplemente debemos modificar la entrada */etc/inetd.conf* como sigue:

```
$>sudo vim /etc/inetd.conf
#/etc/in.fingerd finger daemon
#finger stream tcp nowait root /usr/sbin/in.fingerd in.fingerd
finger stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.fingerd
```

Así conseguimos que se ejecute el comando *tcpd* en lugar del comando *in.fingerd* real. El nombre de la ruta de acceso completa del demonio *finger* se pasa a *tcpd* como un argumento y *tcpd* utiliza este argumento para abrir el demonio real tras haber confirmado que se permite el acceso.

Tendremos que realizar estos cambios para cada uno de los demonios que deseemos proteger.

4.2. Utilizar envoltorios TCP con *xinetd*

xinetd es un reemplazo para *inetd* que están adoptando algunas distribuciones, como Red Hat. Normalmente, *xinetd* tiene una compatibilidad integrada para admitir envoltorios TCP, por lo que lo único que tendremos que hacer es modificar los archivos de configuración del envoltorio TCP (*/etc/hosts.allow* y */etc/hosts.deny*) tal y como se describe en la siguiente sección.

4.3. */etc/hosts.allow* y */etc/hosts.deny*

Los envoltorios TCP utilizan dos archivos de configuración: */etc/hosts.allow* y */etc/hosts.deny*. Estos archivos se utilizan para especificar las reglas de acceso para cada demonio de red protegido con los envoltorios TCP.

Cuando se llama a un envoltorio TCP, lo primero que se hace es obtener la dirección IP del *host* que se está conectando e intenta buscar su nombre de *host* utilizando una búsqueda *DNS* inversa. A continuación, se consulta el archivo */etc/hosts.allow* para comprobar si este *host* en concreto puede acceder al servicio solicitado. Si se encuentra una coincidencia, se permite el acceso y se llama al demonio real. Si no se encuentra ninguna coincidencia en el archivo */etc/hosts.allow*, entonces se procede a consultar el archivo */etc/hosts.deny* para comprobar si este *host* tiene denegado específicamente el acceso. Si se encuentra aquí una coincidencia, se cierra la conexión. Si no se encuentra ninguna coincidencia en ninguno de los dos archivos, se concede el acceso. Esta simple técnica es lo suficientemente eficaz como para cubrir la mayoría de requerimientos de acceso.

La sintaxis de los archivos */etc/hosts.allow* y */etc/hosts.deny* es muy simple. Cada archivo contiene un conjunto de reglas. Cada regla ocupa generalmente una línea pero se puede dividir en varias líneas utilizando una barra invertida al final de la línea.

Cada regla sigue la siguiente forma general:

lista_demonios : *lista_clientes* : *comando_shell*

La siguiente lista explica cada uno de los campos:

- ***lista_demonios***: Es una lista de demonios separados por comas a los que se aplican las reglas. Los demonios se especifican utilizando su nombre de comando base, es decir, el

nombre del programa real del demonio ejecutable que se ejecuta para conseguir el servicio solicitado.

- **lista_clientes:** Es una lista de nombres de *hosts* o direcciones IP separados por comas para los que se aplica la regla.
- **comando_shell:** Es opcional y especifica un comando que se va a ejecutar cuando la regla coincide, algo que se puede utilizar, por ejemplo, para registrar las conexiones entrantes.

lista_demonios y *lista_clientes* pueden contener patrones que nos permitan comparar diversos demonios o *hosts* sin tener que denominarlos explícitamente. Asimismo, podemos utilizar diversos símbolos predefinidos para que las reglas sean más fáciles de leer y de crear.

Vamos a empezar con un simple archivo */etc/hosts.deny* como este:

```
$> cat /etc/hosts.deny
# /etc/hosts.deny
ALL: ALL
```

La primera línea es un comentario. La siguiente línea es una regla que se interpreta como sigue: “Denegar las solicitudes de acceso para todos los servicios de todos los *hosts*”. Si el archivo */etc/hosts.allow* está vacío, esta regla tendrá el efecto de denegar el acceso a todo lo proveniente de todos los *hosts* de Internet, incluyendo el *host* local. Para solucionar este problema, podemos realizar un simple cambio en el archivo:

```
$> sudo vim /etc/hosts.deny
# /etc/hosts.deny
#ALL: ALL
ALL: ALL EXCEPT localhost
```

En realidad esta regla es bastante segura y considerada como un valor predeterminado seguro. Debemos recordar que las reglas en el archivo */etc/hosts.allow* se consultan antes que las del archivo */etc/hosts.deny*, por lo que al agregar reglas a */etc/hosts.allow* podemos sobrescribir esta configuración predeterminada en */etc/hosts.deny*. Por ejemplo, imaginemos que deseamos permitir que todos los *hosts* en Internet tengan acceso al demonio *finger*. Para ello, debemos añadir una regla como la siguiente al archivo */etc/hosts.allow*:

```
$> sudo vim /etc/hosts.allow
# /etc/hosts.allow
in.fingerd: ALL
```

Un uso más común de los envoltorios TCP es restringir el conjunto de *hosts* que pueden acceder al servicio. Los *hosts* se pueden especificar utilizando la dirección IP, el nombre de *host* o algún patrón basado en la dirección o en el nombre de *host*, por ejemplo para especificar un grupo de *host*. Un ejemplo de ello puede ser considerar que el demonio *finger* sólo se encuentre disponible para un pequeño conjunto de *hosts* de confianza. En este caso, modificaríamos el archivo */etc/hosts.allow* de la siguiente manera:

```
$> sudo vim /etc/hosts.allow
# /etc/hosts.allow
in.fingerd: spaghetti.vpasta.com, .vpizza.com, 192.168.1.
```

En este ejemplo hemos elegido permitir solicitudes de *finger* del host denominado *spaghetti.vpasta.com*, así como de cualquier host que se encuentre en el dominio *vpizza.com*, y de cualquier sistema con una dirección IP que empiece con el patrón 192.168.1.

Es importante conocer las reglas que coinciden con el *host* y con la dirección IP en */etc/hosts.allow* y en */etc/hosts.deny* así como la ubicación de los caracteres de punto. Un patrón que empieza con un punto se supone que tiene el nombre del dominio al que tienen que pertenecer los sistemas solicitantes. Un patrón que termina con un punto se supone que especifica un patrón de dirección IP.

5. Cortafuegos: filtrado de paquetes IP

Aunque podemos utilizar envoltorios TCP para restringir el conjunto de *hosts* que pueden establecer conexiones con determinados servicios en una máquina, muchas veces es mejor ejercer un control mucho más detallado sobre los paquetes que pueden o no introducirse en un determinado sistema. También existen envoltorios TCP que sólo funcionan con servicios configurados utilizando *inetd* o *xinetd*; algunos servicios son independientes y proporcionan sus propias opciones de control de acceso y otros servicios no implantan ningún tipo de control de acceso, por lo que es necesario proporcionar otro nivel de protección si deseamos controlar las conexiones efectuadas a dichos servicios. Hoy en día es muy común que los usuarios de Internet se protejan a sí mismos ante amenazas de los ataques basados en la red utilizando una técnica denominada filtrado IP. El filtrado IP implica inspeccionar el núcleo cada vez que se transmite o se recibe un paquete de red y decidir si se le puede permitir el paso, si hay que denegárselo, o hay que modificarlo de alguna manera antes de permitir que pase. Normalmente el filtrado IP se conoce como cortafuegos ya que al filtrar cuidadosamente los paquetes que se introducen o que dejan una máquina, estamos creando un cortafuegos entre el sistema y el resto de Internet. El filtrado IP no nos protege frente a los ataques de virus o troyanos o frente al defecto de una aplicación, pero puede protegernos ante muchas formas de ataques basados en la red, como determinados tipos de ataques de *DoS* y paquetes IP fraudulentos, es decir, paquetes que se marcan como entrantes de un determinado sistema cuando en realidad provienen de otro. El filtrado IP también proporciona una capa adicional de control de acceso que evita que usuarios no deseados intenten obtener acceso al sistema.

Para que funcione el filtrado IP, tenemos que saber cuáles son los paquetes permitidos y cuáles son los paquetes denegados. Normalmente, la decisión de filtrar un paquete se basa en los encabezados del paquete, que contienen información como las direcciones IP de origen y destino, el tipo de protocolo (TCP, UDP, etc) y los números del puerto origen y destino, los cuales identifican el servicio para el que se destina el paquete. Los distintos servicios de red utilizan diferentes protocolos y números de puerto; por ejemplo, la mayoría de servidores web reciben solicitudes TCP en el puerto 80.

Algunas veces, una simple inspección del encabezado del paquete no es suficiente para conseguir la realización de una determinada tarea de filtrado, por lo que tendremos que inspeccionar e interpretar los datos reales transportados dentro del paquete. A veces esta técnica se conoce como inspección de paquetes de datos (*SPI*, *Stateful Packet Inspection*) porque el paquete se considera dentro del contexto de una conexión de red en curso en lugar de considerarse individualmente. Por ejemplo, podríamos permitir que los usuarios que se encuentran dentro de nuestra red utilicen servidores *ftp* que se encuentran fuera de nuestra red. *ftp* es un protocolo complejo que utiliza una conexión TCP para enviar comandos al servidor, pero utiliza otra para transferir datos. Lamentablemente, la especificación *ftp* no obliga a que se utilice un determinado número de puerto para transferir datos, por lo que el cliente y el servidor tienen que negociar los números de puerto utilizando la sesión de comandos. Sin la inspección de paquetes de datos, permitir transferencias *ftp* requeriría permitir las conexiones TCP para puertos arbitrarios. *SPI* resuelve este problema al interpretar la negociación del número de

puerto entre el cliente y el servidor y permitir que pasen automáticamente los paquetes TCP en el puerto negociado.

El filtrado IP se implanta a través del núcleo de Linux, que contiene un código para inspeccionar cada paquete que se recibe y se transmite aplicando reglas de filtrado que determinan el destino del paquete. Las reglas se configuran utilizando una herramienta de configuración en el espacio de usuario que acepta argumentos desde la línea de órdenes y los traduce en especificaciones de filtro que se guardan y se utilizan como reglas por el núcleo.

En Linux han existido tres generaciones de filtrado IP basado en el núcleo y cada una ha tenido su propio mecanismo de configuración. La primera generación se denominaba *ipfw* (**IP FireWall**) y proporcionaba una opción básica de filtrado, pero a veces era algo inflexible e ineficiente para configuraciones complejas. Actualmente se utiliza en raras ocasiones. La segunda generación de filtrado, denominada cadenas IP (*ipchains*), mejoró extraordinariamente a *ipfw* y sigue utilizándose con frecuencia. La última generación de filtrado se denomina *netfilter/iptables*. *netfilter* es el componente del núcleo e *iptables* es la herramienta de configuración del espacio del usuario; dichos términos se intercambian indistintamente. *netfilter* no es sólo más flexible de configurar que los primeros filtros sino que también es extensible.

6. Otros elementos útiles para asegurar nuestro sistema

Aparte de los elementos vistos en secciones anteriores, cabe destacar que existen otros que nos van a permitir mejorar aun más la seguridad del sistema, algunos de estos son herramientas de trabajo que todo administrador debe conocer y manejar y otros son concejos de configuraciones o demonios que no se deben habilitar en los sistemas para evitar ponerlos en riesgo.

6.1. El servicio *finger* para descubrimiento de usuarios

El servicio *finger* utiliza el puerto 79 TCP y ha sido una de las principales fuentes de problemas de los sistemas operativos Unix. Este protocolo proporciona información, demasiado detallada, de los usuarios existentes en una máquina, estén o no conectados en el momento de acceder al servicio.

Para acceder a la información de usuario, se suele utilizar *finger* desde un cliente, pasándole como argumento un nombre de máquina precedido del símbolo @ y, opcionalmente, un nombre de usuario. Primero *finger* devolverá los datos generales de los usuarios conectados en ese momento a la máquina, y después informará con más detalle del usuario especificado como parámetro, esté o no conectado. Este servicio proporciona mucha información delicada como: nombres de usuarios, hábitos de conexión, cuentas inactivas, etc.

Debido a que *finger* es una herramienta que revela demasiada información es obligatorio que como administradores de sistemas lo desactivemos a lo inmediato, pues muchas distribuciones de Linux vienen por defecto con este servicio habilitado.

6.2. Fortaleza de contraseñas con *crack*

Si por alguna razón consideramos que los usuarios del sistema no están imponiendo contraseñas difíciles de averiguar, podríamos ejecutar periódicamente un programa rompedor de contraseñas con el fin de asegurarnos si las contraseñas de nuestros usuarios están siendo seguras.

Los programas rompedores de contraseñas operan sobre una idea simple: prueban cada palabra del diccionario, y después las variaciones de estas palabras, encriptando cada una y

comprobándola frente a la contraseña encriptada. Si obtienen una que concuerde, entonces saben cuál es la contraseña.

El comando *crack* nos permite comprobar la fortaleza de las contraseñas que existen dentro de nuestro sistema, ubicadas en */etc/shadow*. Cabe destacar que generalmente el proceso consume mucho tiempo de CPU, pero sólo sabremos si un atacante podría lograr romper las contraseñas de los usuarios de nuestro sistema si somos nosotros mismos los que las rompemos primero.

Una vez que nos encontramos con algún usuario que ha establecido una contraseña débil, lo mejor que podemos hacer es inhabilitar la cuenta inmediatamente y luego ponernos en contacto con él para comunicarle lo sucedido.

Cabe destacar que existen otras herramientas alternativas al comando *crack* (ver tabla 8.1) que también nos permiten romper contraseñas débiles, algunas de estas herramientas son:

Herramienta	Descripción
<i>Jhon the Ripper</i>	Herramienta de auditoria de propósito general para sistemas Windows y Linux. Utiliza algoritmos propios y admite un gran conjunto de reglas y opciones.
<i>Lard</i>	Herramienta de auditoria de contraseñas para Linux y otras versiones de Unix. Esta herramienta se destaca por ser lo suficientemente pequeña como para caber en un disquete, lo que resulta de mucha utilidad para auditar equipos no necesariamente conectados en red.
<i>Xcrack</i>	Este es un <i>script</i> en <i>Perl</i> que nos permite romper contraseñas de Linux. Ejecuta un cifrado completo del archivo de diccionarios y esta pensado para entornos donde se espera que existan contraseñas excepcionalmente débiles.

Tabla 8.1: Herramientas alternativas al comando *crack*

6.3. Comprobación proactiva de contraseñas

Debido a que son los usuarios los que establecen la gran mayoría de las contraseñas existentes en los sistemas, es necesario implementar un mecanismo que nos permita de alguna manera definir las reglas que deben cumplir las contraseñas para que estas no sean consideradas como débiles. Ha estos mecanismos se les conoce como mecanismos de comprobación proactiva de contraseñas, los cuales hacen que se eliminen las contraseñas débiles establecidas por los usuarios antes de que estas sean enviadas y almacenadas en la base de datos de contraseñas (*/etc/shadow*) del sistema.

La idea se basa en que cuando un usuario crea una contraseña, ésta se compara en primer lugar con una lista de palabras y luego con una serie de reglas ambas establecidas por el administrador del sistema. Si la contraseña que intenta establecer el usuario no cumple con los requisitos de este proceso, entonces automáticamente el usuario es obligado por el sistema a formular otra contraseña.

En Linux básicamente predominan tres mecanismos de comprobación proactiva de contraseñas los cuales son: *passwd+*, *anlpasswd* y *npasswd*. En las siguientes secciones vamos a hacer una breve descripción de cada uno de ellos.

6.3.1. Comprobador proactivo de contraseñas: *passwd+*

Es un comprobador proactivo de contraseñas que provee un lenguaje de *script* para chequear las contraseñas.

Por ejemplo:

- `test length("$p")<6`
Si la contraseña tiene menos de seis caracteres, rechazarla.
- `test infile("/usr/dict/words", "$p")`
Si la contraseña es una palabra que pertenece a la lista de palabras definidas como no validas, rechazarla.
- `test !inprog("spell", "$p", "$p")`
Si la contraseña no está en la salida del programa *spell*, rechazarla.

6.3.2. Comprobador proactivo de contraseñas: *anlpasswd*

Esta herramienta, escrita en código *Perl*, ejecuta una serie de verificaciones sobre las contraseñas cuando estas se establecen.

Las reglas predeterminadas que implementa son:

- La contraseña debe estar formada por números con espacios.
- La contraseña debe estar formada por letras en mayúsculas y en minúsculas con espacios.
- La contraseña debe estar formada por letras escritas todas en mayúsculas o todas en minúsculas.
- La contraseña debe estar formada únicamente por números.
- La contraseña debe estar formada por la primera letra en mayúscula y el resto de la contraseña por números.
- La contraseña puede estar formada por algunas o todas las combinaciones de las reglas anteriores.

6.3.3. Comprobador proactivo de contraseñas: *npasswd*

npasswd es una herramienta que pretende sustituir al comando *passwd* de cualquier sistema Linux. Es una herramienta recomendable ya que obliga a los usuarios a establecer contraseñas aceptables, rechazando contraseñas débiles formadas con palabras de diccionario, con palabras que incluyen el *login* o la *password*, con palabras de longitud demasiado cortas, etc.

Algunas características que presenta *npasswd* es la capacidad de someter a las contraseñas que quieren establecer los usuarios a estrictas pruebas de capacidad de adivinación, además su distribución cuenta con un conjunto de herramientas de desarrollo para poder ampliar o incorporar *npasswd* con otras aplicaciones.

Algunas de las reglas que presenta esta herramienta son las siguientes:

- Permite decidir cuál va a ser la longitud mínima de todas las contraseñas.
- Obligar a los usuarios a mezclar mayúsculas y minúsculas.
- No aceptar palabras simples como una letra repetida.

- Rechazar contraseñas que contengan el nombre de la máquina u otra información relacionada con ella.
- Comprueba si la contraseña elegida contiene el nombre de la cuenta del usuario o el nombre de apellido del mismo.
- Comprueba si la contraseña está incluida en algunos de los diccionarios, incluido el del sistema.

Como regla general, a pesar del uso de *passwd+*, *npasswd* o *anlpasswd*, cualquier administrador deberá ejecutar con cierta periodicidad algún programa adivinador, tipo *crack*, para comprobar que a los usuarios no se les ha permitido establecer ninguna contraseña débil.

6.4. Mapeo de puertos con *nmap*

El mapeo de puertos es una técnica ampliamente utilizada para identificar los servicios que ofrecen los sistemas. Con esta información, los administradores son capaces de tomar decisiones respecto a llevar a cabo bloqueos de servicios y puertos que son innecesarios en el sistema y que pueden permitir a un atacante llevar a cabo una intrusión.

La aplicación por excelencia para realizar mapeo de puertos es *nmap* (*Network MAPper*), esta es una aplicación de código abierto que sirve para efectuar rastreo de puertos TCP y UDP, además es utilizada para evaluar la seguridad de sistemas informáticos, así como para descubrir servicios o servidores dentro de una red informática.

Algunas características de *nmap* son:

- Permite llevar a cabo descubrimiento de equipos, es decir, identifica que máquinas están disponibles dentro de una red.
- Identifica los puertos que se encuentran abiertos o disponibles en una máquina objetivo.
- Determina los servicios que se encuentran disponibles en una máquina objetivo.
- Determinar el tipo de sistema operativo y la versión del mismo que esta siendo utilizado en una máquina objetivo, esta técnica es también conocida como *fingerprinting*.

nmap es considerada como una de las muchas herramientas imprescindibles para todo administrador de sistema, es utilizada por los mismos para llevar a cabo pruebas de penetración y tareas de seguridad informática.

Los administradores de sistemas utilizan *nmap* para buscar fallas en sus propias redes y equipos conectados a la misma, o bien para detectar máquinas que no cumplen con ciertos requisitos mínimos de seguridad.

6.5. Sistemas de detección de intrusos

La detección de ataques e intrusiones parte de la idea que un atacante es capaz de violar nuestra política de seguridad, atacando parcial o totalmente los recursos de una red, con el objetivo final de obtener un acceso con privilegios de administrador.

Los mecanismos para la detección de ataques e intrusiones tratan de encontrar y reportar la actividad maliciosa en la red, pudiendo llegar a reaccionar adecuadamente ante un ataque.

En la mayoría de los casos es deseable la capacidad de identificar el ataque exacto que se está produciendo, de forma que sea posible detener el ataque y recuperarse del mismo. En otras situaciones, sólo será posible detectar e informar de la actividad sospechosa que se ha encontrado, ante la imposibilidad de conocer lo que ha sucedido realmente.

Generalmente, la detección de ataques trabajará con la premisa de que nos encontramos en la peor de las situaciones, suponiendo que el atacante ha obtenido un acceso al sistema y que es capaz de utilizarlo o modificar sus recursos.

Los elementos más destacables dentro de la categoría de mecanismos para la detección de ataques e intrusiones son los sistemas de detección de intrusos *IDS (Intrusion Detection System)*.

La intrusión consiste en la secuencia de pasos realizados por el atacante para violar una determinada política de seguridad. La existencia de una política de seguridad, en la que se contemplan una serie de acciones deshonestas que hay que prevenir, es un requisito clave para la intrusión. Es decir, la violación sólo se podrá detectar cuando las acciones observadas puedan ser comparadas con el conjunto de reglas definidas en la política de seguridad.

La detección de intrusiones es el proceso de identificación y respuesta ante las actividades ilícitas observadas contra uno o varios recursos de una red.

6.5.1. Detección de ataques en red con *snort*

snort es una completa herramienta de seguridad basada en código abierto para la creación de sistemas de detección de intrusos en entornos de red. Cuenta con una gran popularidad entre la comunidad de administradores de redes y servicios. Gracias a su capacidad para la captura y registro de paquetes en redes TCP/IP, *snort* puede ser utilizado para implementar desde un simple *sniffer* de paquetes para la monitorización del tráfico de una pequeña red, hasta un completo sistema de detección de intrusos en tiempo real.

Mediante un mecanismo adicional de alertas y generación de ficheros de registro, *snort* ofrece un amplio abanico de posibilidades para la recepción de alertas en tiempo real acerca de los ataques y las intrusiones detectadas.

Como monitor de red, *snort* se comporta como una auténtica aspiradora, de ahí su nombre, de datagramas IP, ofreciendo diferentes posibilidades en cuanto a su tratamiento. Desde actuar como un simple monitor de red pasivo que se encarga de detectar el tráfico maligno que circula por la red, hasta la posibilidad de enviar a servidores de ficheros de registro o servidores de base de datos todo el tráfico capturado.

6.6. Advertencias ante paquetes con *bugs*

Otro problema de seguridad al cual nos tenemos que enfrentar los administradores de sistemas es al problema de las aplicaciones con errores (*bugs*). Es por eso que debemos tener especial cuidado al momento de instalar o actualizar cualquier aplicación, ya que si esta posee algún *bug* debemos determinar si dicho *bug* pondrá en peligro la seguridad de nuestro sistema.

Debido a que en los sistemas Linux existen una gran variedad de aplicaciones llevar a cabo esta tarea no sería nada fácil, es por esto que las distribuciones que hacen uso del sistema de gestión de paquetes *APT (Advanced Packaging Tool)*, cuentan con una herramienta llamada *apt-listbugs*, la cual permite recibir mensajes de advertencia cuando se quiere realizar una

instalación o actualización de algún paquete que posee errores críticos. La idea es que después de instalar la herramienta, automáticamente cada vez que se ejecute el comando *apt-get install* o *apt-get upgrade*, y exista algún error en algún paquete que se dispone a ser instalado en el sistema, se reciba un mensaje y el poder de decisión para proseguir con la instalación o actualización, o en caso de no querer correr el riesgo de instalar el paquete con el *bug* entonces poder cancelar la instalación.

La tarea del administrador al encontrarse frente a un paquete que presente errores críticos será investigar sobre cual es el error crítico que presenta el paquete y en función de eso realizar un análisis de los riesgos que implica la instalación del mismo dentro del sistema.

6.7. Auditando vulnerabilidades con *nessus*

nessus es un programa de escaneo de vulnerabilidades existentes en diversos sistemas operativos. Consiste en *nessusd*, el *daemon nessus*, que realiza el escaneo en el sistema objetivo, y *nessus*, el cliente (basado en modo consola o modo gráfico) que muestra el avance y reporte de los escaneos. Desde el modo consola *nessus* puede ser programado para hacer escaneos automáticos junto con el administrador regular de procesos *cron*.

En operación normal, *nessus* comienza escaneando los puertos con *nmap* o con su propio escaneador de puertos para buscar puertos abiertos y después intentar varios *exploits* para atacarlo. Las pruebas de vulnerabilidad, disponibles como una larga lista de *plugins*, son escritos en *NASL* (*Nessus Attack Scripting Language*), un lenguaje scripting optimizado para interacciones personalizadas en redes.

Opcionalmente, los resultados del escaneo pueden ser exportados en reportes en varios formatos, como texto plano, XML, HTML, y LaTeX. Los resultados también pueden ser guardados en una base de conocimiento para referencia en futuros escaneos de vulnerabilidades.

Algunas de las pruebas de vulnerabilidades de *nessus* son tan efectivas que pueden causar que los servicios o el propio sistema operativo se corrompan y caigan.

TEMA 9: COPIAS DE SEGURIDAD Y RECUPERACIÓN

Objetivos

- Caracterizar los beneficios que aporta el mecanismo de copias de seguridad a los sistemas informáticos.
- Comprender la relación estricta que existe entre las copias de seguridad y la recuperación ante el fallo de un sistema informático.
- Estudiar y poder aplicar algunas de las herramientas más importantes que existen en los sistemas Linux para la implementación de copias de seguridad.
- Describir algunas técnicas de copias de seguridad sobre elementos críticos existentes en el sistema, para su posterior recuperación.

Contenido

1. Introducción a las copias de seguridad
 - 1.1. Modelos de almacén de datos
 - 1.2. Medios de almacenamiento
 - 1.3. Administrar un almacén de datos
2. Concejos a tener en cuenta al realizar copias de seguridad
3. Planificación de las copias de seguridad
4. Copias de seguridad completas
5. Copias de seguridad incrementales
6. Aplicando compresión a las copias de seguridad
 - 6.1 Problemas que presenta comprimir las copias de seguridad
7. Otras herramientas para realizar copias de seguridad en Linux
 - 7.1. Copias de seguridad utilizando *dd*
 - 7.2. Copias de seguridad utilizando *dump*
 - 7.3. Copias de seguridad utilizando *cpio*
 - 7.4. Copias de seguridad utilizando *afio*
 - 7.5. Copias de seguridad utilizando *rsync*
 - 7.6. Copias de seguridad utilizando *amanda*
8. ¿Qué hacer en caso de emergencia?
9. Revisar y recuperar sistemas de archivos
10. Recuperación del súper bloque
11. *MBR* dañado, infectado o corrupto

Bibliografía

Básica

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada”, Editorial Anaya Multimedia, 2006.
- Iñaki Alegría Loinaz, Roberto Cortiñas Rodríguez, Aitzol Ezeiza Ramos, “Linux Administración del sistema y la red”, Editorial Prentice Hall, 2005.

Complementaria:

- Jack Tackett Jr. y David Gunter, “Linux Tercera Edición, Edición Especial”, Editorial Prentice Hall, 1998.
- Copia de seguridad.
http://es.wikipedia.org/wiki/Copia_de_seguridad

Una de las funciones más importantes de la administración de sistemas es la recuperación de la información tras una pérdida de la misma. Los motivos por los que se producen pérdidas de información pueden ser diversos: ataques, errores de usuario, accidentes, etc. Para recuperar la información será preciso haber realizado con anterioridad un trabajo preventivo: la creación y el correcto almacenamiento de copias de seguridad.

1. Introducción a las copias de seguridad

Hacer una copia de seguridad o una copia de respaldo (*backup*) se refiere a llevar a cabo una copia de todos o algunos de los datos existentes en un sistema informático, de tal forma que estas copias adicionales de dichos datos puedan restaurar todo el sistema o algunos archivos del mismo después de una pérdida de información.

Las copias de seguridad resultan esencialmente útiles por dos razones:

1. Permiten restaurar una máquina a un estado operacional después de un desastre, este tipo de copias de seguridad son conocidas como *copias de seguridad del sistema*.
2. Permiten restaurar un pequeño número de archivos después de que estos hayan sido borrados o dañados accidentalmente, este tipo de copias de seguridad son conocidas como *copias de seguridad de datos*.

Normalmente las copias de seguridad se suelen hacer en cintas magnéticas, pero dependiendo de lo que se trate de almacenar en la copia de seguridad también es posible utilizar disquetes, CD, DVD, discos ZIP, JAZ o magnético-ópticos, pendrives, discos duros o pueden realizarse sobre un centro de respaldo remoto propio o vía internet.

Las copias de seguridad en un sistema informático tienen por objetivo mantener cierta capacidad de recuperación de la información ante posibles pérdidas. Esta capacidad puede llegar a ser algo muy importante, incluso crítico, para cualquier empresa. Por ejemplo, existen casos de empresas que han llegado a desaparecer ante la imposibilidad de recuperar sus sistemas al estado anterior a que se produjese un incidente de seguridad grave.

Las copias de seguridad juegan un papel importante para los administradores de sistemas ya que estas suelen ser utilizadas por los mismos como la última línea de defensa contra pérdida de datos, y se convierten por lo tanto en el último recurso que estos suelen utilizar.

1.1. Modelos de almacén de datos

Cualquier estrategia de copia de seguridad empieza con el concepto de almacén de datos. Los datos de la copia de seguridad deben ser almacenados de alguna manera y probablemente deben ser organizados bajo algún criterio. Esto puede ser tan simple como una hoja de papel con una lista de todas las cintas de copias de seguridad junto con las fechas en las que estas fueron creadas y las fechas en las que se hizo uso de las mismas para restaurar el sistema. O incluso podemos hacer uso de un sofisticado programa con un índice computarizado, con un catálogo o con una base de datos relacional. Cada uno de los distintos almacenes de datos tiene sus ventajas. Esto está muy relacionado con el esquema de rotación de copia de seguridad elegido.

A continuación se detallan algunos de los modelos de almacén de datos más ampliamente utilizados en la actualidad.

Desestructurado

Un almacén desestructurado podría simplemente ser una pila de disquetes, cintas, CD-R, etc. con una mínima información sobre qué ha sido copiado y cuándo. Éste es el modelo más fácil de implementar, pero ofrece pocas garantías de recuperación de datos.

Completa + incremental

Un almacén completo + incremental propone hacer más factible el almacenamiento de varias copias de la misma fuente de datos. En primer lugar se realiza una copia de seguridad completa del sistema. Más tarde se realiza una copia de seguridad incremental, es decir, sólo con los ficheros que se hayan modificado desde la última copia de seguridad. Restaurar un sistema completamente a un cierto punto en el tiempo requiere localizar una copia de seguridad completa y todas las incrementales posteriores realizadas hasta el instante que se desea restaurar. Los inconvenientes que presenta este modelo de almacén de datos son tener que tratar con grandes series de copias incrementales y contar con un gran espacio de almacenaje.

Espejo + diferencial

Un almacén de tipo espejo + diferencial es muy similar al almacén completo + incremental. La diferencia radica en que en vez de hacer una copia completa seguida de un conjunto de copias incrementales, este modelo ofrece un espejo que refleja el estado del sistema a partir de la última copia y un historial de copias diferenciales. Una ventaja de este modelo es que sólo requiere una copia de seguridad completa inicial. Cada copia diferencial es inmediatamente añadida al espejo y los ficheros que son reemplazados son movidos a una copia incremental inversa. Una copia diferencial puede sustituir a otra copia diferencial más antigua sobre la misma copia total.

Protección continua de datos

Este modelo toma un paso más lejos y en vez de realizar copias de seguridad periódicas, el sistema inmediatamente registra cada cambio en las copias de seguridad que se estén generando. La idea es que cada cambio que se efectuó en el sistema será escrito de forma inmediata en la copia de seguridad.

1.2. Medios de almacenamiento

A pesar del modelo de almacén que se decida utilizar, los datos de las copias de seguridad tienen que ser guardados en unos medios de almacenaje, es por eso que a continuación se detallan los medios de almacenaje más conocidos.

Cinta magnética

La cinta magnética es, por mucho, considerada como el medio de almacenaje más común usado para volcar datos almacenados, copias de seguridad, archivadores, etc. La cinta magnética ha tenido comúnmente un orden de magnitud mejor de proporción capacidad/precio comparado con los discos duros, pero últimamente las proporciones capacidad/precio entre discos duros y cintas magnéticas son cada vez más cercanas. Existen multitudes de formatos de cintas magnéticas, algunos de los cuales son específicos de mercados como unidades principales o a rangos de ordenadores particulares. La cinta magnética es un medio de acceso secuencial, por ello aunque el tiempo de acceso es lento, la tasa de escritura y lectura continua de datos suele ser muy rápida. Algunas unidades de cinta son incluso más rápidas que algunos discos duros actuales.

Disco duro

La proporción capacidad/precio de los discos duros ha sido rápidamente mejorada en los últimos años. Esto los ha convertido en medios de almacenaje muy competitivos con respecto a las cintas magnéticas. La principal ventaja de los discos duros es la gran capacidad y el corto tiempo de acceso que estos poseen.

Disco óptico

Un CD-R puede ser usado como un mecanismo de copia de seguridad. Una ventaja de los CDs es que estos pueden almacenar hasta 700 MB de datos en 12 cm (4.75"). Además de que pueden ser utilizados en cualquier máquina que posea una unidad de CD-ROM. Otro de los formatos que está siendo ampliamente utilizado para llevar a cabo copias de seguridad son los DVD. Muchos de los formatos de discos ópticos son de tipo escritura única, aunque también los hay de re-escritura, lo que los convierte en medios útiles para fines de almacenamiento ya que los datos una vez que se han escrito no pueden ser modificados.

Disquetes

Durante la década de los ochenta y principios de los noventa, muchas personas y usuarios de ordenadores personales asociaban las copias de seguridad con los disquetes. La baja capacidad de almacenamiento de datos de los disquetes los ha convertido actualmente en un medio de almacenamiento obsoleto y en desuso.

Dispositivos de memoria no volátil

También conocidos como memorias flash, llaves USB, compact flash, smart media, sticks de memoria, tarjetas Secure Digital, etc., estos dispositivos se están convirtiendo en medios de almacenamiento útiles para copias de seguridad debido a sus bajos costos, sus grandes capacidades de almacenamiento de datos y su fácil manejabilidad.

Servicios remotos de copia de seguridad

Debido a la amplia extensión que ha tenido Internet en las últimas décadas y a las grandes velocidades que se pueden conseguir con las redes actuales, los servicios de copia de seguridad remota han ganado gran popularidad. Copias de seguridad vía internet a una localización remota, puede protegernos ante hechos diversos como incendios o destrucciones de sistemas locales de copia de seguridad. Uno de los principales inconvenientes que presentan los servicios remotos de copia de seguridad es que la velocidad de conexión a Internet suele ser menor que la velocidad de los dispositivos de almacenamiento de datos, algo que se llega a convertir en un verdadero problema cuando la cantidad de información a respaldar es demasiado grande.

1.3. Administrar un almacén de datos

A pesar del modelo de almacén de datos o del medio de almacenamiento utilizado en una copia de seguridad, el sistema necesita encontrar un nivel entre accesibilidad, seguridad y coste.

A continuación se estudian los distintos niveles que alcanzan los sistemas de copias de seguridad según sea el lugar en el que estos se decidan colocar.

En línea

El almacenamiento en línea es típicamente el más accesible de los tipos de almacenamiento de datos. Un buen ejemplo de este podría ser un largo vector de discos en el cual se están almacenando continuamente las copias de seguridad. Este tipo de almacenamiento es muy

conveniente y rápido, pero presenta el problema de que es relativamente costoso; además se encuentra típicamente localizado en cercana proximidad al sistema que ha sido copiado. Esta proximidad representa un problema en caso de cualquier tipo de desastre. Adicionalmente, el almacenamiento en línea es vulnerable de ser borrado o de ser sobrescrito, incluso por accidente, o causado por un virus en el sistema.

Cerca de línea

El almacenamiento cercano en línea es típicamente menos costoso y más accesible que el almacenamiento en línea. Un buen ejemplo de este sería una biblioteca de cintas. Un dispositivo mecánico está involucrado en mover unidades de almacenamiento desde el almacén donde están guardadas todas las cintas hasta el lector donde estas son leídas para restaurar el sistema o escritas para almacenar una nueva copia de seguridad.

Fuera de línea

Un almacenamiento fuera de línea es similar al cercano en línea, exceptuando que el almacenamiento fuera de línea requiere de la interacción por parte de una persona para conseguir que los medios de almacenamiento estén disponibles. Esto puede ser tan simple como almacenar las cintas de copias de seguridad en un armario de ficheros y que el administrador las saque del armario para que estas sean utilizadas para restaurar el sistema o bien utilizadas para guardar en ellas una nueva copia de seguridad.

Cámara fuera del lugar

Para protegerse contra cualquier tipo de desastres u otro tipo de problemas, muchas empresas eligen mandar los medios de copia de seguridad a una cámara fuera del lugar de trabajo, es decir en una ubicación distinta a la del sistema original. La cámara puede ser tan simple como la oficina ubicada en la casa del administrador del sistema o puede ser tan sofisticada como un búnker insensible, con alta seguridad y con temperatura controlada que cuenta con equipos para almacenar las copias de seguridad.

Centro de recuperación de datos

Existen empresas que realizan contratos con agencias de recuperación de datos, estas agencias les permiten realizar a las empresas copias de seguridad de sus datos a través de Internet, así como también recuperar las copias de seguridad que estas han almacenado en sus servidores cuando la empresa lo desee. Además, estas agencias ofrecen garantías de seguridad y confidencialidad sobre los datos y permiten ampliar las capacidades de almacenamiento contratadas de forma ilimitada.

2. Concejos a tener en cuenta al realizar copias de seguridad

Lo que se pretende con la realización de copias de seguridad es poder restaurar archivos individuales o sistemas de archivos completos. Todo lo que se haga en relación al tema de copias de seguridad deberá tener muy en cuenta este propósito.

Algunos concejos útiles al realizar copias de seguridad son:

- Preparar un programa de copias de seguridad que detalle los archivos que deben protegerse, la frecuencia con que se van a realizar las copias de seguridad y la forma en que se van a restaurar los archivos.
- Informar a todos los usuarios sobre este tema y la forma en la que estos pueden pedir la restauración de sus archivos.

- Atenerse estrictamente al plan diseñado.
- Verificar siempre las copias de seguridad. La comprobación puede consistir, por ejemplo, en la lectura de una tabla de contenidos sobre los medios utilizados después de haberlos almacenado, o en la restauración de un archivo seleccionado al azar. Debemos recordar que siempre cabe la posibilidad de que el medio de almacenamiento de las copias de seguridad, disco, cintas, CD-R, etc. pueda estar defectuoso.
- Hacer copias de seguridad de forma que los archivos puedan restaurarse en cualquier lugar del sistema de archivos o en otro sistema informático. Para esto último podemos hacer uso de utilidades de copia de seguridad que permitan que dichas copias puedan ser usadas en otros sistemas informáticos Linux o Unix.
- No olvidarse de etiquetar todos los medios de cintas, discos, CD-R, etc., que se utilicen para almacenar las copias de seguridad. Si un mismo proceso requiere de varios medios de almacenamiento, debemos asegurarnos de que todos ellos son enumerados secuencialmente y fechados. De esta forma podremos localizar fácilmente el archivo o archivos que necesitemos restaurar.
- Planificar siempre pensando que va a pasar lo peor. Debemos tener copias de los archivos del sistema, de forma que éste pueda restaurarse en un plazo razonable de tiempo.
- Almacenar las cintas, discos, CD-R, etc. que contienen las copias de seguridad en un lugar distinto al ocupado por el sistema.
- Minimizar el efecto o la carga que genera el realizar copias de seguridad. ¿realizar una operación de copia de seguridad aumenta la carga del sistema? ¿supondrá una incomodidad o molestia notable por los usuarios? Además si se realiza una copia de seguridad de una base de datos activa, probablemente no se harán copias de seguridad de los archivos modificados durante el proceso, lo cual puede representar un inconveniente o una cuestión importante. Por ello, es preferible realizar copias de seguridad cuando el sistema se encuentra relativamente tranquilo, a fin de que la copia de seguridad sea lo más completa posible.
- Planificar una comprobación periódica de los procedimientos de copia de seguridad para asegurarnos de que estos se ajustan a nuestras necesidades.
- Si se decide hacer la copia de seguridad a un segundo disco duro interno, es recomendable mantener al menos el disco sin montar cuando no se esté utilizando para que, en caso de que eliminemos accidentalmente uno o más sistemas de archivos, podamos utilizar la copia de seguridad disponible.
- Evaluar la importancia relativa de la seguridad y capacidad de recuperación de los datos frente al coste y conveniencia del medio de copia de seguridad que sea elegido así como su uso.

Estos forman parte de algunos de los consejos que son de gran utilidad para los administradores de sistemas. Cabe destacar que a medida que se va adquiriendo experiencia en el proceso de copias de seguridad surgirán nuevos elementos particulares de acuerdo a las necesidades de cada administrador.

3. Planificación de las copias de seguridad

Es importante preparar un programa de copias de seguridad que se ajuste a nuestras necesidades y que permita restaurar copias recientes de archivos. Una vez que hayamos seleccionado el programa, debemos procurar ajustarnos a él.

Lo ideal sería poder restaurar cualquier archivo en cualquier momento. Desgraciadamente, esto no es posible, aunque sí deberíamos poder restaurar archivos diariamente. Para ello, debemos utilizar una combinación de copias de seguridad completa e incremental.

En esta sección vamos a estudiar un poco más a fondo estos dos modelos de almacén de datos, el modelo de copias de seguridad completas y el modelo de copias de seguridad incrementales, los cuales son los más utilizados, y por ende los más importantes, en los sistemas Linux.

En un modelo de copia de seguridad completo se guarda toda la información del sistema de archivos. En ocasiones se suele utilizar una versión reducida: la copia completa de la información modificable. En este último caso se guarda toda la información de los usuarios, pero no la que no se modifica, como el propio sistema, las aplicaciones, etc. En cualquier caso, la realización de copias de seguridad completas exige mucho tiempo y un soporte de gran capacidad.

El modelo de copias de seguridad incrementales, es especialmente útil si existen numerosos usuarios dentro del sistema o son llevados a cabo frecuentes cambios en la configuración del mismo. En este modelo de copias de seguridad se debe realizar una copia de seguridad completa sólo una vez al mes. A continuación, cada semana, sólo se deben copiar los archivos que han cambiado con respecto a la semana anterior. Asimismo, cada noche, se deben hacer copias de seguridad sólo de los archivos que han cambiado con respecto a las veinticuatro horas anteriores. Los modelos de copias de seguridad incrementales son muy eficientes ya que hacen copias de seguridad en pequeños pasos, utilizan menos medios de almacenamiento de datos, las copias de seguridad generadas diaria y semanalmente son mucho más cortas y más fáciles de ejecutar y como mucho proporcionan copias de seguridad que cuentan con un día de antigüedad. Por ejemplo, Supongamos que accidentalmente fue eliminado todo el sistema de archivos de un determinado ordenador; para restaurarlo desde una copia de seguridad que utiliza un modelo de copias de seguridad incrementales lo único que se debe hacer es seguir los siguientes pasos:

1. Restaurar la copia de seguridad mensual más reciente. Por ejemplo, si el sistema fue eliminado el diecisiete de julio, se deberá restaurar la copia de seguridad completa del primero de julio. El sistema reflejará ahora el estado de los archivos cuando se efectuó la copia de seguridad del primero de julio.
2. Restaurar cada una de las copias de seguridad semanales realizadas a lo largo del mes. Para el caso del ejemplo planteado, se deberán restaurar las dos copias semanales efectuadas el siete de julio y el catorce de julio. Al restaurar cada una de las copias de seguridad semanales, todos los archivos cambiados durante esas semanas se verán actualizados.
3. Restaurar cada una de las copias de seguridad diarias durante la semana pasada, es decir, desde la última copia semanal. Para el caso del ejemplo planteado, se deberán restaurar las copias de seguridad diarias del quince y del dieciséis de julio. Ahora el sistema está igual que cuando se ejecutó la copia de seguridad diaria el dieciséis de julio; no se han perdido archivos de más de un día.

Para asegurar que la información se recupera correctamente, será preciso disponer de la última copia de seguridad completa junto con todas las incrementales que posteriormente se

hayan realizado. De este modo, dentro de la política de seguridad se debe decidir la frecuencia con la que se realizarán las copias completas e incrementales de seguridad.

Una característica importante que deben tener normalmente las copias de seguridad es la capacidad de seleccionar archivos individuales existentes dentro de la copia de seguridad para su restauración. De este modo, si un archivo o un conjunto de archivos se eliminan accidentalmente, es posible simplemente restaurar los archivos necesarios sin tener que realizar una restauración completa del sistema. Sin embargo, dependiendo de cual sea el sistema de copias de seguridad que se elija, esta tarea puede ser muy fácil o realmente difícil.

En la siguiente sección vamos a estudiar el uso de las herramientas básicas que nos proporcionan los sistemas Linux para llevar a cabo copias de seguridad tanto completas como incrementales.

4. Copias de seguridad completas

Los sistemas Linux ponen a nuestra disposición un comando de uso común para la generación de copias de seguridad completas. Este es el comando *tar*, con él es posible almacenar de modo conjunto varios ficheros o directorios en un solo archivo. Además de almacenar los ficheros o directorios que contienen la información, este comando también guardará su estructura, es decir, el subárbol formado por los subdirectorios y ficheros del directorio que vayamos a respaldar. De este modo, al recuperar la información se mantendrá la estructura original, independientemente de si se restaura en el directorio original o en otra ubicación.

Para crear una copia de seguridad utilizando el comando *tar* se debe hacer uso de la opción *c* (*create*) y para restaurar la copia de seguridad se debe hacer uso de la opción *x* (*extract*). Teniendo esto en cuenta, se pueden realizar copias de seguridad completas mediante el comando *tar* de la siguiente manera:

```
$> tar cvf /media/HDDexterno/backup.tar /
```

De este modo se consigue una copia de seguridad completa de todo el sistema (/), en el archivo *backup.tar*. Si se desea, el fichero *backup.tar* puede ser, más adelante, almacenado en un disco óptico utilizando una grabadora de CD.

Para la recuperación de los archivos contenidos en la copia de seguridad se deberá utilizar un comando semejante al anterior, pero cabe destacar que previamente se deberá copiar en el directorio adecuado del disco duro del sistema el fichero *backup.tar*.

```
$> cp /media/HDDexterno/backup.tar /  
$> cd /  
/$> tar xvf backup.tar
```

Con esto se logra recuperar todos los ficheros existentes dentro de la copia de seguridad. Sin embargo, en ocasiones sólo se desea recuperar un fichero o un subconjunto de ficheros existentes dentro de la copia de seguridad. En este caso, el primer paso que se debe hacer es obtener los nombres de los ficheros a recuperar. Con el siguiente comando se mostrará en pantalla el contenido de los ficheros dentro de *backup.tar*.

```
/$> tar tvf backup.tar
```

Una vez que se han obtenido los nombres de los archivos que se quieren restaurar, estos pueden ser recuperados con el siguiente comando:

```
/$> tar xvf backup.tar <ListaDeFicherosARestaurarSeparadosPorEspacio>
```

5. Copias de seguridad incrementales

Las copias de seguridad incrementales, tal y como se han descrito anteriormente, son una forma extraordinaria de mantener actualizadas las copias de seguridad de nuestro sistema. Por ejemplo, se pueden realizar copias nocturnas sólo de los archivos que hayan cambiado las últimas veinticuatro horas, así mismo se pueden realizar copias de seguridad semanales de todos los archivos que hayan cambiado en la última semana y también se pueden realizar copias de seguridad mensuales de todo el sistema.

Para realizar copias de seguridad incrementales en Linux también se puede hacer uso del comando *tar* junto con otro comando, el comando *find*, el cual permite conocer que ficheros o directorios han sido modificados las últimas veinticuatro horas o la última semana. Pero además del comando *tar* los sistemas Linux proporcionan otra serie de herramientas que permiten llevar a cabo copias de seguridad más sofisticadas o mejor elaboradas, estas herramientas serán analizadas en la sección denominada *otras herramientas para realizar copias de seguridad en Linux*.

El primer paso en la creación de una copia de seguridad incremental es producir una lista con los ficheros o directorios que hayan cambiado desde una determinada cantidad de tiempo, para realizar esta tarea, como se indicó anteriormente, se deberá hacer uso del comando *find*.

Por ejemplo, para generar una lista con todos los archivos modificados en las últimas veinticuatro horas, podemos utilizar el siguiente comando:

```
$> sudo find / -mtime -1 \! -type d -print > /media/HDDexterno/Lista.diaria
```

El primer argumento de *find* es el directorio desde donde se va a empezar la búsqueda (aquí, /, el directorio raíz). La opción *-mtime -1* le indica a *find* que localice todos los archivos que hayan cambiado en las últimas veinticuatro horas (1 día).

La parte *\! -type d* es un poco complicada, pero recorta de la salida alguna información innecesaria. Le indica a *find* que excluya directorios de la lista de archivos resultante. El signo *!* es un operador de negación, que le indica que sean excluidos todos los archivos del tipo *d* (*directory*); se debe colocar delante de este operador una barra invertida ** porque de lo contrario la *shell* lo interpretaría como un carácter especial.

La opción *-print* imprime en la salida estándar todos los nombres de los archivos que coinciden con la búsqueda. Y por último la salida estándar es redirigida a un archivo para su uso posterior. Asimismo, para localizar todos los archivos que hayan cambiado durante la semana pasada, se deberá utilizar el siguiente comando:

```
$> sudo find / -mtime -7 \! -type d -print > /media/HDDexterno/Lista.semanal
```

Cabe destacar que si *find* es utilizado de esta manera, recorrerá todos los sistemas de archivos que se encuentren montados en el ordenador. Por ejemplo, si se tiene montado un CD-ROM, *find* también intentará localizar todos los archivos o directorios dentro del CD-ROM que cumplan con la condición de búsqueda, los cuales probablemente no se necesiten incluir dentro

de la copia de seguridad. Para limitar la búsqueda de *find* al sistema de archivos local se debe utilizar la opción *-xdev*.

Ahora ya se ha producido una lista de archivos para realizar la copia de seguridad. Anteriormente, cuando se utilizó el comando *tar*, se tuvo que especificar los archivos a almacenar en la línea de comandos. Sin embargo, esta lista de archivos, generada por *find*, puede ser demasiado larga para una sola línea de comandos, la cual normalmente se limita a 2048 caracteres aproximadamente, y la propia lista se encuentra dentro de un archivo.

Para especificar un archivo que contiene una lista de archivos para una copia de seguridad utilizando el comando *tar*, se deberá hacer uso de la opción *-T*. Para poder utilizar esta opción, se debe utilizar una sintaxis alternativa para el comando *tar* en la que todas las opciones son especificadas explícitamente con guiones. Por ejemplo, para crear una copia de seguridad de los archivos que se encuentran en */media/HDDexterno/Lista.diaria* se debe utilizar el comando:

```
$> tar -cv -T /media/HDDexterno/Lista.diaria -f /media/HDDexterno/backup.tar
```

También se puede seguir el mismo mecanismo para generar las copias de seguridad semanales y mensuales.

6. Aplicando compresión a las copias de seguridad

Las copias de seguridad pueden generar problemas en los dispositivos de almacenamiento debido a su gran tamaño. Para minimizar este problema se pueden utilizar programas de compresión. Los comandos que más se utilizan con este fin en el mundo Linux son *gzip* y *gunzip*. El primero permitirá comprimir la información y el segundo permitirá descomprimirla.

Siguiendo los ejemplos anteriores de generación de copias de seguridad mediante el comando *tar*, para comprimir las copias de seguridad se utilizará el siguiente comando:

```
$> gzip /media/HDDexterno/backup.tar
```

De este modo, se obtiene un fichero de nombre *backup.tar.gz* y que contiene la misma información que contenía el fichero *backup.tar* original, pero lo que cambia es que el tamaño de *backup.tar.gz* es más reducido que el del fichero original. La extensión *.gz* hace referencia a que el fichero se encuentra comprimido.

En los sistemas Linux actuales es posible realizar la compresión al mismo tiempo que se realiza la copia de seguridad. Para ello bastará con utilizar la opción *z* en el comando *tar*. Así para realizar copias completas de seguridad y al mismo tiempo de todo el sistema comprimido se puede utilizar el siguiente comando:

```
$> tar cvfz media/HDDexterno/backup.tgz /
```

La extensión *.tgz* es equivalente a *tar.gz* y además ambas son intercambiables, es decir, el primero se puede descomprimir haciendo uso directamente del comando *tar* y el segundo aunque no es comprimido directamente con *tar*, sino que es comprimido con *gzip*, también puede ser descomprimido directamente con *tar*.

Por ejemplo, podemos realizar una copia de seguridad del directorio */boot* utilizando el comando *tar* y efectuando la compresión directamente con *tar*:

```
$> tar cvfz ~/boot.tgz /boot
$> ls -l ~/
-rw-r--r-- 1 gateway gateway 18842991 2008-05-15 14:08 boot.tgz
$> tar xvfz ~/boot.tgz
$> ls -l
drwxr-xr-x 3 gateway gateway 4096 2008-05-06 02:11 boot
```

De igual manera, podemos realizar una copia de seguridad del directorio */boot* utilizando el comando *tar* pero sin efectuar la compresión directamente con *tar*, luego comprimimos la copia de seguridad con el comando *gzip*, y comprobamos que es posible descomprimirla directamente utilizando el comando *tar*:

```
$> tar cvf ~/boot.tar /boot
$> ls -l ~/
-rw-r--r-- 1 gateway gateway 20152320 2008-05-15 13:56 boot.tar
$> gzip ~/boot.tar
$> ls -l ~/
-rw-r--r-- 1 gateway gateway 18843000 2008-05-15 13:56 boot.tar.gz
$> tar xvfz ~/boot.tar.gz
$> ls -l
drwxr-xr-x 3 gateway gateway 4096 2008-05-06 02:11 boot
```

En las últimas versiones de Linux ha surgido un nuevo formato para la compresión, de nombre *bzip2*. Aunque su capacidad de compresión es mayor que la de *gzip*, este último continúa siendo el más utilizado, ya que es más rápido y estándar. El nombre del comando correspondiente a este nuevo formato también es *bzip2*, y se integra en el comando *tar* mediante el uso de la opción *j*.

6.1 Problemas que presenta comprimir las copias de seguridad

Cabe destacar que existen muchos argumentos, tanto a favor como en contra, de la compresión de archivos *tar* cuando se crean copias de seguridad. El problema general es que ni el comando *tar* ni los comandos de compresión *gzip* o *bzip2* son particularmente tolerante a fallos, independientemente de su conveniencia. Aunque la compresión de *gzip* o *bzip2* puede reducir extraordinariamente la cantidad de medios de copia de seguridad requeridos para guardar un archivo, comprimir archivos *tar* completos hace que la copia de seguridad sea propensa a una pérdida completa de datos, si uno de los bloques del archivo está dañado.

Aunque *tar* no proporciona mucha protección frente a los datos dañados dentro de un archivo, si existe un daño mínimo dentro de un archivo *.tar*, normalmente se podrá recuperar la mayoría de los archivos guardados sin problemas, o en el peor de los casos, solo los archivos anteriores al archivo dañado. Aunque está lejos de ser algo perfecto, es mejor que perder toda la copia de seguridad.

Una mejor solución al problema de los errores de compresión es utilizar una herramienta distinta a *tar* para crear copias de seguridad. Los sistemas Linux ponen a disposición de los administradores de sistemas distintas opciones para la creación de copias de seguridad tolerantes a fallos. Dentro de estas se encuentra *cpio* la cual es una utilidad de almacenamiento que empaqueta archivos, similar a *tar*. Sin embargo, debido al método de almacenamiento más

simple de *cpio*, este se recupera limpiamente de una corrupción de datos en un archivo. Sin embargo, sigue sin controlar bien los errores en archivos comprimidos con *gzip* o *bzip2*.

La mejor solución parece ser utilizar herramientas como *afio*. La cual es una herramienta que admite copias de seguridad de múltiples volúmenes y es similar, en algunos aspectos, a *cpio*. Sin embargo, *afio* incluye la compresión y es más fiable, ya que comprime cada archivo individualmente, lo que significa que si se dañan los datos de un archivo, el daño se puede aislar a archivos individuales en lugar de a toda la copia de seguridad.

7. Otras herramientas para realizar copias de seguridad en Linux

En esta sección estudiaremos otras herramientas útiles para crear copias de seguridad en los sistemas Linux, como lo son: *dd*, *dump*, *cpio*, *afio*, *rsync* y *amanda*. Algunas de estas herramientas son más avanzadas que otras. La decisión de utilizar una u otra deberá ser tomada por el administrador del sistema, el cual deberá evaluar cual o cuales son las que mejor se adaptan a sus necesidades.

7.1. Copias de seguridad utilizando *dd*

Para llevar a cabo copias de seguridad completas de todo el sistema, Linux nos proporciona el comando *dd* (*duplicate disk*), el cual es una utilidad originaria de Unix y debería formar parte de la *caja de herramientas* de todo administrador de sistemas Linux.

La forma de trabajo del comando *dd* para llevar a cabo copias de seguridad completas es la siguiente:

Para realizar una copia de seguridad sin aplicarle compresión utilizando *dd*, podemos hacer uso del siguiente comando:

```
$> dd if=/dev/hda1 of=/dev/sdb1
```

De este modo se consigue una copia de seguridad de todo el sistema ubicado en */dev/hda1* (*/dev/hda1* = la primera partición del primer disco duro IDE en el ordenador) y se almacena dicha copia de seguridad en */dev/sdb1* (para el caso del ejemplo una memoria USB de 16GB de almacenamiento).

Para realizar la restauración de los archivos y directorios de la copia de seguridad generada por *dd* en el ejemplo anterior, podemos hacer uso del siguiente comando:

```
$> dd if=/dev/sdb1 of=/dev/hda1
```

Para realizar una copia de seguridad aplicándole compresión utilizando *dd* y *gzip*, podemos hacer uso del siguiente comando:

```
$> dd if=/dev/hda1 | gzip > /media/HDDexterno/sistema.gz
```

De este modo se consigue una copia de seguridad de todo el sistema ubicado en */dev/hda1* y se almacena el archivo que contiene la copia de seguridad en */media/HDDexterno* (para el caso del ejemplo un disco duro externo). Pero además comprimimos todo utilizando *gzip*.

Para realizar la restauración de los archivos y directorios de la copia de seguridad generada por *dd* y *gzip* en el ejemplo anterior, podemos hacer uso del siguiente comando:

```
$> gzip -dc /media/HDDexterno/sistema.gz | dd of=/dev/hda1
```

Cabe destacar que existen dos principales inconvenientes al realizar copias de seguridad con el comando *dd*. El primero es que es demasiado lento y el segundo es que guarda el espacio no ocupado por los datos que se encuentran en la partición sobre la cual efectuamos la copia de seguridad.

7.2. Copias de seguridad utilizando *dump*

Esta herramienta tiene una mayor potencia que *tar* y está más orientada a la realización de copias de seguridad. Sin embargo, no aparece por defecto en todas las distribuciones de Linux y solo se puede utilizar en las particiones de tipo *ext2* y *ext3*.

Las copias realizadas con el comando *dump* serán completas o incrementales en función del parámetro utilizado. Los parámetros más utilizados son:

- **Nivel:** Mediante un número se indica si se desea realizar una copia de seguridad completa (0 ó 1) o incremental (2-9). El nivel, cinco por ejemplo, señala que sólo se deben guardar los archivos modificados desde las copias incrementales de nivel inferior, tres y cuatro en el ejemplo. La información necesaria para ello queda almacenada en el fichero */etc/dumpdates*.
- **Ubicación destino:** Se utiliza para indicar el dispositivo o fichero en el que se almacenará la copia de seguridad, por ejemplo, el dispositivo */media/HDDexterno*.
- **Directorio-raíz:** Se utiliza para especificar el sistema de ficheros o subconjunto de ficheros que se desea proteger. Los valores habituales son */* y */home*.

Se pueden probar distintas combinaciones de niveles con el fin de encontrar el mejor equilibrio entre la seguridad y el tiempo de recuperación. Una opción recomendable es que las copias de seguridad generadas semanalmente sean de nivel tres y las copias de seguridad generadas diariamente sean de nivel cuatro.

Para almacenar todos los ficheros modificados desde el último *backup* completo (segundo nivel), se puede utilizar el siguiente comando:

```
$> dump -2 /media/HDDexterno /
```

Las restauraciones de las copias de seguridad generadas se pueden realizar utilizando el comando *restore*. Un comando típico para una recuperación completa puede ser:

```
$> cd /  
$> restore -rf /media/HDDexterno
```

También se ofrece un modo de recuperación interactivo, que permite recuperar de modo independiente algunos ficheros o directorios. Para ello se deberá indicar la opción *-i* y con la operación *add* se deberán marcar los elementos que hay que recuperar. Por último, la opción *extract* realizará la restauración.

7.3. Copias de seguridad utilizando *cpio*

cpio (**CoPy In/Out**) es una utilidad que permite copiar archivos a o desde una copia de seguridad *cpio*, que no es más que un fichero que almacena otros archivos e información sobre ellos (permisos, nombres, propietario, etc.). La copia de seguridad puede ser almacenada en un disco, otro archivo, una cinta o incluso una tubería, mientras que los ficheros a copiar pueden ser archivos normales, pero también dispositivos o sistemas de archivo completos.

En la tabla 9.1 se muestran las opciones de *cpio* más utilizadas; la sintaxis de esta orden es bastante más confusa que la de *tar* debido a la interpretación de lo que *cpio* entiende por *dentro* y *fuera*: copiar *fuera* indica generar la copia de seguridad en la salida estándar (que con toda probabilidad desearemos redireccionar), mientras que copiar *dentro* es lo contrario, es decir, extraer archivos de la entrada estándar (que también es seguro que desearemos redireccionarla).

Opción	Acción que realiza
o	Copiar fuera (<i>out</i>).
i	Copiar dentro (<i>in</i>).
m	Conservar la fecha y la hora de los archivos.
t	Crea tabla de contenidos.
A	Añade ficheros a una copia de seguridad existente.
v	Modo <i>verbose</i> .

Tabla 9.1: Opciones básicas del comando *cpio*

Por ejemplo, si deseamos generar una copia de seguridad de los ficheros existentes en el directorio */home* y almacenar la copia de seguridad en */tmp/backup.cpio* podemos utilizar la siguiente sintaxis:

```
$> find /home | cpio -o > /tmp/backup.cpio
```

Como podemos ver, *cpio* lee la entrada estándar esperando los nombres de ficheros a guardar, por lo que es conveniente utilizarlo tras una tubería pasándole esos nombres de archivo. Además, hemos de redirigir su salida al nombre que queramos asignarle a la copia de seguridad, ya que de lo contrario se mostraría el resultado en la salida estándar (lo que evidentemente no es muy utilizado para realizar *backups*). Podemos fijarnos también en que estamos usando el comando *find* en lugar de un simple *ls*: esto es debido a que *ls* mostraría sólo el nombre de cada fichero en lugar de su ruta completa, por lo que *cpio* buscaría dichos ficheros a partir del directorio actual.

Una vez creada la copia de seguridad quizás resulte interesante chequear su contenido, con la opción *-t*. Por ejemplo, la siguiente orden mostrará en pantalla el contenido de */tmp/backup.cpio*:

```
$> cpio -t < /tmp/backup.cpio
```

Igual que para hacer copias de seguridad de ficheros o directorios hemos de pasarle a *cpio* la ruta de los mismos, para extraerlos también lo debemos hacer así. Si no indicamos lo contrario el comando *cpio -i* extraerá todos los archivos y directorios de la copia de seguridad, pero si sólo nos interesan algunos de ellos podemos especificar su nombre de la siguiente forma:

```
$> cd /home
$> echo "/home/toni/hola.txt" | cpio -i < /tmp/backup.cpio
```

7.4. Copias de seguridad utilizando *afio*

afio es otra de las herramientas que Linux pone a nuestra disposición para generar copias de seguridad. Constituye una mejor forma de tratar archivos con formato *cpio*. Generalmente es una herramienta que suele ser mucho más rápida que *cpio*. *afio* proporciona múltiples opciones para realizar copias de seguridad sobre cintas magnéticas y se recupera bastante bien de la corrupción en los datos de entrada. Admite archivos multi-volumen durante su operación interactiva. *afio* permite crear copias de seguridad que son mucho más seguras que las copias de seguridad generadas con *tar* o con *cpio*.

Un ejemplo típico de la forma de uso del comando *afio* es el siguiente:

```
$> find / -depth -print0 | afio -px -0a /media/HDDexterno
```

afio es la herramienta más utilizada para generar copias de seguridad en cintas magnéticas.

7.5. Copias de seguridad utilizando *rsync*

rsync es una poderosa implementación de un pequeño y maravilloso algoritmo. Su principal poder es la habilidad de replicar eficientemente un sistema de archivos. Usando *rsync*, es fácil configurar un sistema que mantendrá una copia actualizada de un sistema de archivos usando un arreglo flexible de protocolos de red, tales como *NFS*, *SMB* o *ssh*. La segunda gran característica que consigue este sistema de copias de seguridad es la capacidad de archivar viejas copias de seguridad que han sido modificadas o eliminadas.

En resumen, este sistema usa un ordenador de pocas prestaciones con un sistema Linux, integrado a muchos discos duros baratos y con un pequeño *script* que llama a *rsync* (ver figura 9.1). Cuando se hace una copia de seguridad, le decimos a *rsync* que cree un directorio llamado *YY-DD-MM* como lugar para almacenar los cambios incrementales. Seguidamente, *rsync* examina los servidores de los que hacemos copias de seguridad de los cambios. Si un archivo ha cambiado, se copia la versión vieja al directorio incremental, y luego sobrescribe el archivo en el directorio principal de copias de seguridad (ver figura 9.2).

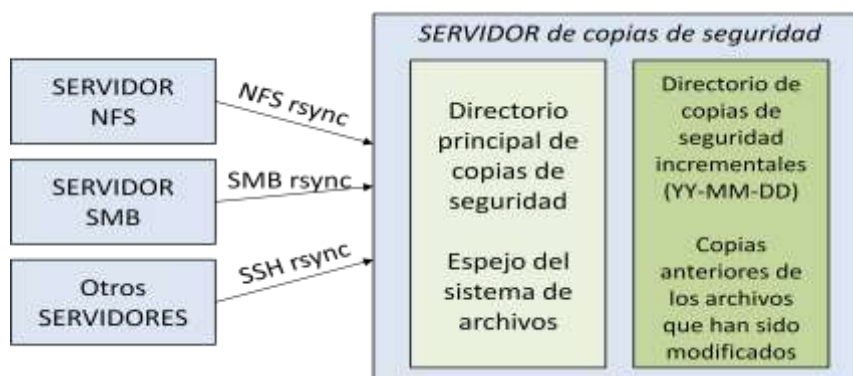


Figura 9.1: Estructura de un servidor de copias de seguridad con *rsync*

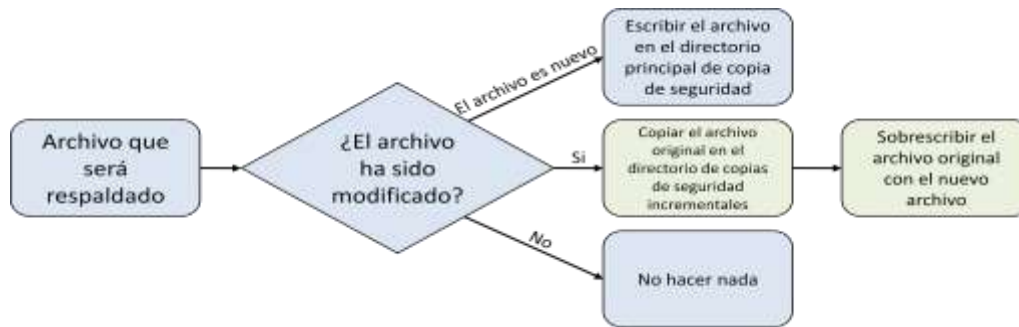


Figura 9.2: Copia de seguridad de un archivo utilizando *rsync*

La forma básica para trabajar con *rsync* viene dado por un script. En realidad sólo hay un comando:

```
rsync --force --ignore-errors --delete --delete-excluded --exclude-
from=ListaDeArchivosAExcluir --backup --backup-dir=`date +%Y-%m-%d` -av
```

Los parámetros y opciones son:

- **--backup:** Crea copias de seguridad de los archivos antes de sobrescribirlos.
- **--backup-dir=`date +%Y-%m-%d`:** Crea un directorio de copia de seguridad, de nombre año-mes-día, para esas copias de seguridad.
- **-av:** Modo de archivo y modo detallado.

Este script se puede programar para que se ejecute cada noche usando el administrador regular de procesos en segundo plano (*cron*) de Linux. Por ejemplo, para conseguir que el script se ejecute cada noche a las 11:00 PM, se debe utilizar el comando *contrab -e*, y luego escribir lo siguiente: `0 23 * * * /RutaDeUbicacionDelScript`

7.6. Copias de seguridad utilizando *amanda*

amanda (*Advanced Maryland Automatic Network Disk Archiver*) es un sistema de copias de seguridad desarrollado para entornos Unix y Linux, que permite realizar copias de seguridad en red, utilizando un único servidor de copias de seguridad centralizado para salvaguardar información contenida en diferentes máquinas de una red local. Aunque *amanda* funciona sólo en sistemas Unix y Linux, permite también hacer copias de seguridad de máquinas Windows 95/98/NT mediante *SAMBA*.

La utilidad *amanda* fue desarrollada en la universidad de Maryland y actualmente es mantenida por la comunidad Open Source. *amanda* gestiona el proceso de hilvanar una combinación de copias de seguridad completas e incrementales a partir de series de clientes de red (servidores y estaciones de trabajo) a un disco de un *host* de cinta. Estas series de datos se escriben en cintas. Cuando se instala correctamente, *amanda* realiza su tarea de forma automática. Una tarea *cron*, sobre cada *host* de cinta, controla que la cinta correcta está cargada en cada unidad de cinta, que todos los clientes están accesibles y que hay suficiente espacio disponible para almacenar las copias de seguridad. Esta tarea se suele hacer durante el día para que los errores encontrados se puedan enviar a tiempo a una lista de operadores de copia y administradores de sistema, con el fin de que estos cambien las cintas y corrijan otros errores. Otra tarea *cron*, realizada generalmente a altas horas de la noche, ejecuta las copias de seguridad, conectando desde el *host* de cinta con cada uno de los clientes. *amanda* cuenta con funciones que ejecutan conexiones en paralelo y que controlan su propio uso de ancho de banda.

Un módulo, conocido como el planificador, es el que determina qué niveles de incrementales hay que ejecutar en cada sistema de archivos de cada *host*.

amanda utiliza *dump* (opcionalmente) o *tar* para llevar a cabo las copias de seguridad. Luego es posible, con un poco de automatización de comandos como *dd* y *mt*, extraer los archivos de *amanda* por medio de herramientas convencionales. Normalmente se emplea el comando *amrecover* para realizar una restauración de alguna copia de seguridad.

8. ¿Qué hacer en caso de emergencia?

No es difícil cometer un error como *root* que nos cause problemas reales en nuestro sistema, como por ejemplo, no poder iniciar la sesión de nuevo o perder archivos importantes, algo que suele sucederles especialmente a los administradores de sistemas más novatos. Pero casi todos los administradores de sistemas aprendemos nuestras buenas lecciones por la vía difícil: viéndonos obligados a recuperar el sistema a partir de una emergencia real. En esta sección proporcionaremos algunas sugerencias sobre lo que tenemos que hacer cuando sucedan dificultades.

Debemos adoptar siempre medidas preventivas que reduzcan el impacto de dichas emergencias. Por ejemplo, debemos crear una copia de seguridad de todos los archivos importantes del sistema, si no de todo el sistema. Las copias de seguridad son vitales para recuperarnos ante cualquier problema; no debemos dejar perder muchas semanas de duras tareas realizadas para la configuración de nuestro sistema Linux, por no haber realizado ninguna copia de seguridad.

Asimismo, debemos asegurarnos de escribir notas sobre la configuración de nuestro sistema, con las entradas de las tablas de particiones, los tamaños y tipos de particiones y los sistemas de archivos que estas contienen. Si por alguna razón se dañase la tabla de particiones, la solución al problema podría ser simplemente cuestión de volver a ejecutar *fdisk*, pero esta solución sólo nos ayudará si somos capaces de recordar cuál era la apariencia de nuestra tabla de particiones.

De hecho, no es mala idea hacer una copia de seguridad de las tablas de particiones de nuestro sistema. El programa *sfdisk* es una herramienta muy útil para ver, guardar y manipular datos de nuestras tablas de particiones. Por ejemplo, podemos capturar y guardar estos datos en un archivo con un comando como el siguiente:

```
$> sfdisk -d /dev/hda > /particiones.lista
```

De esta forma volcaremos las tablas de particiones del primer disco duro de nuestro ordenador y las guardaremos en el archivo */particiones.lista* (o como queramos llamarlo). Esta salida puede ser leída por nosotros y por el comando *sfdisk*.

Si tenemos que restaurar una tabla de particiones, podemos editar el archivo */particiones.lista*, eliminar toda aquella información de particiones que no deseemos restaurar y volver a montar la tabla de particiones de la siguiente forma:

```
$> sfdisk /dev/hda < /particiones.lista
```

Evidentemente, para que funcione cualquiera de estas medidas necesitaremos un método para arrancar el sistema y acceder a nuestros archivos, o recuperar todo el sistema a partir de las copias de seguridad, en una emergencia, algo que se consigue mejor con un *disco de emergencia* o *disco raíz*. Normalmente, se trata de un disquete o CD-ROM de arranque que contiene lo suficiente para que el sistema Linux pueda recuperar sistemas de archivos y realizar

tareas de reparación. Existen también CD-ROM muy avanzados, como *Knoppix* (<http://www.knopper.net/knoppix/index-en.html>), que arrancan el sistema con un escritorio gráfico, un explorador web y cualquier otro elemento para trabajar con comodidad. Cualquiera de estos tipos pueden resultarnos útiles si necesitamos recuperar el sistema cuando se produce un fallo.

9. Revisar y recuperar sistemas de archivos

En ocasiones es necesario revisar los sistemas de archivos de Linux y repararlos si aparecen errores o si experimentamos pérdidas de datos. Dichos errores se producen normalmente tras el cierre repentino del sistema o por una falta de suministro eléctrico, consiguiendo que el núcleo sea incapaz de sincronizar la memoria caché del búfer del sistema de archivos con el contenido en el disco. La mayoría de veces dichos errores son relativamente menores. Sin embargo, si el sistema se cerrase durante la escritura de un archivo grande, el archivo puede haberse perdido y los bloques asociados con él pueden haberse marcado como *en uso* cuando, realmente, no existe ninguna entrada del archivo que se corresponda con ellos. En otros casos, los errores pueden originarse a raíz de una escritura de datos accidental directamente en el dispositivo del disco duro, como */dev/hda*, o en una de las particiones, como */dev/hda2*.

Para revisar los sistemas de archivos y corregir cualquier problema se utiliza el comando *fsck*. *fsck* es una interfaz para un *fsck.tipo* de tipo específico de sistema de archivos, como lo es *fsck.ext2* para los sistemas de archivos *Second Extended*. *fsck.ext2* es un vínculo simbólico a *e2fsck*, pudiéndose ejecutar cualquiera de ellos directamente, si no se tiene instalada la interfaz *fsck*.

El uso de *fsck* es muy sencillo; el formato del comando es:

```
$> fsck -t <tipo> <dispositivo>
```

Siendo *tipo*, el tipo de sistema de archivos a reparar y *dispositivo*, el dispositivo (partición de unidad, disquete, etc.) en el que reside el sistema de archivos. Por ejemplo, para revisar el sistema de archivos *ext3* en */dev/hda2* podemos utilizar el siguiente comando:

```
$> fsck -t ext3 /dev/hda2
fsck 1.34 (25-Jul-2003)
/dev/hda2 is mounted. Do you really want to continue (y/n)? y

/dev/hda2 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

Free blocks count wrong for group 3 (3331, counted=3396) . FIXED
Free blocks count wrong for group 4 (1983, counted=2597) . FIXED
Free blocks count wrong (29643, counted=30341) . FIXED
Inode bitmap differences: -8280 . FIXED
Free inodes count wrong for group #4 (1405, counted=1406) . FIXED
Free inodes count wrong (34522, counted=34523) . FIXED

/dev/hda2: ***** FILE SYSTEM WAS MODIFIED *****
/dev/hda2: ***** REBOOT LINUX *****
/dev/hda2: 13285/47808 files, 160875/191216 blocks
```


En primer lugar, el sistema nos solicita una confirmación antes de revisar el sistema de archivos montado. Si se encuentra cualquier error y se corrige durante el uso de *fsck*, tendremos que reiniciar el sistema, si el sistema de archivos está montado. La razón es que los cambios realizados por *fsck* no se propagan al conocimiento interno del sistema en el diseño del sistema de archivos. En general no es recomendable revisar sistemas de archivos montados.

Como podemos comprobar con el ejemplo anterior, *fsck* encuentra y corrige diversos problemas y como el sistema de archivos del ejemplo estaba montado, el sistema nos informa de que debemos reiniciar la máquina.

¿Cómo podemos revisar los sistemas de archivos sin montarlos? Excepto para el sistema de archivos principal, podemos desmontar simplemente cualquier sistema de archivos antes de ejecutar *fsck* sobre ellos. Sin embargo, si el sistema de archivos principal no se puede desmontar mientras se está ejecutando el sistema. Una forma de revisar el sistema de archivos principal mientras está desmontado es utilizar un disquete o CD-ROM de arranque como el que vimos en la sección anterior. De esta forma, el sistema de archivos principal se encuentra en el CD-ROM de arranque, el sistema de archivos principal de nuestro disco duro permanece desmontado y podemos revisar el sistema de archivos principal del disco duro desde ahí.

Muchos sistemas Linux revisan automáticamente los sistemas de archivos en el momento del arranque. Al hacerlo, el sistema normalmente monta inicialmente el sistema de archivos principal como de sólo lectura, ejecuta *fsck* para revisarlo y ejecuta el comando:

```
mount -w -o remount /
```

La opción *-o remount* vuelve a montar el sistema de archivos con los nuevos parámetros, la opción *-w* (equivalente a *-o rw*) monta el sistema de archivos como de lectura-escritura. El resultado es que el sistema de archivos principal se vuelve a montar con un acceso de lectura-escritura.

Podemos especificar opciones al *fsck* específico del tipo. La mayoría de tipos admiten la opción *-a*, que confirma automáticamente cualquier solicitud de línea de comandos que pueda mostrar *fsck.tipo*, la opción *-c*, que efectúa una revisión de bloques dañados y *-v*, que imprime información por la salida estándar durante la operación de revisión. Estas opciones deben de ser especificadas al comando *fsck* tras el argumento *-t tipo*.

No todos los tipos de sistemas de archivos admitidos por Linux tienen una variante *fsck* disponible. Para los sistemas de archivos *Second* y *Third Extended*, los sistemas de archivos *JFS* de *reiser* y los sistemas de archivos *Minix*, no deberíamos tener ningún problema en encontrar una versión de *fsck*.

El uso de *fsck* no significa ni mucho menos que se capturen y reparen todos los errores de los sistemas de archivos, pero sí los problemas más comunes. Si eliminamos un archivo importante, no existe un método fácil para recuperarlo (*fsck* no podrá hacerlo). Es por esto que debemos asegurarnos de realizar copias de seguridad y siempre utilizar *rm -i*, para que se nos consulte antes de eliminar algún archivo.

10. Recuperación del súper bloque

Es posible que se dañe un sistema de archivos de forma que no se pueda montar, algo que normalmente es el resultado de un daño en el súper bloque del sistema de archivos, que guarda información sobre el sistema de archivos como un todo. Si el súper bloque está dañado, el sistema no podrá acceder al sistema de archivos y cualquier intento de montarlo fallará, probablemente con un error parecido a *no se puede leer el súper bloque*.

Debido a la importancia del súper bloque, el sistema de archivos mantiene copias de seguridad de él a intervalos en el sistema de archivos. Los sistemas de archivos *ext2* y *ext3* están divididos en *grupos de bloques*, teniendo cada grupo (de forma predeterminada) 8192 bloques. Por consiguiente, existen copias de seguridad del súper bloque en desplazamientos de los bloques 8193, 16385 (es decir, $8192 \times 2 + 1$), 24577 (es decir, $8192 \times 3 + 1$), etc. Si utilizamos el sistema de archivos *ext2* o *ext3*, podemos comprobar si el sistema de archivos tiene grupos de 8192 bloques con el siguiente comando:

```
$> dumpe2fs /dev/hda2 | more
```

Evidentemente, este comando sólo funciona cuando el bloque maestro está intacto. Este comando imprimirá una gran cantidad de información sobre el sistema de archivos y debe aparecer algo parecido a lo siguiente:

```
Blocks per group: 8192
```

Si el comando nos muestra otro desplazamiento, debemos utilizar los desplazamientos de cómputo para las copias del súper bloque, mencionadas anteriormente.

Si no podemos montar el sistema de archivos debido a problemas con el súper bloque, es muy probable que también falle el comando *fsck.tipo*. Pero podemos indicarle al comando *fsck.tipo* que utilice una de las copias del súper bloque en su lugar, para reparar el sistema de archivos. Por ejemplo:

```
$> fsck -t ext3 -f -b <offset> <device>
```

Siendo *<offset>* el desplazamiento del bloque a una copia del súper bloque (normalmente es 8193). El parámetro *-f* se utiliza para obligar a revisar el sistema de archivos; cuando se utilizan copias de seguridad del súper bloque, el sistema de archivos puede parecer que está *limpio*, en cuyo caso no se necesita ninguna revisión, *-f* sobrescribe este comportamiento. Por ejemplo, para reparar el sistema de archivos que se encuentra en */dev/hda2* con un súper bloque dañado, podemos utilizar el siguiente comando:

```
$> fsck -t ext3 -f -b 8193 /dev/hda2
```

Las copias del súper bloque nos salvan el día. Los comandos anteriores se pueden ejecutar desde un disquete o CD-ROM de arranque y seguramente nos permitirán montar de nuevo nuestro sistema de archivos.

Ahora que los sistemas de ficheros transaccionales como *ext3*, *reiserfs* y *jfs* se incluyen en la mayoría de distribuciones de Linux de forma predeterminada, es muy poco probable que tengamos que utilizar los mecanismos de recuperación del súper bloque que acabamos de estudiar. Gracias a *journal*, que es un registro mantenido internamente por el sistema de archivos con todos los cambios realizados, los sistemas de archivos modernos son muy resistentes a los daños del súper bloque. Aún así, puede ocurrir, por lo que nunca esta de más saber cómo recuperarse fácilmente sin tener que volver a restaurar todo el sistema de archivos.

11. MBR dañado, infectado o corrupto

El *MBR* (*Master Boot Record*) es el primer sector físico de un dispositivo de almacenamiento de datos, como por ejemplo, un disco duro. Es también conocido como sector cero o sector de arranque principal (*bootsector*).

Si el registro de inicio maestro queda dañado este puede repararse. Generalmente, si se produce un daño en el registro de inicio maestro este no tiene por qué afectar a otros sistemas de archivos o datos del resto del dispositivo de almacenamiento de datos.

La prevención puede hacer que esta tarea sea muy sencilla. En otras palabras, al igual que ocurre con muchas otras emergencias, una rutina de preparación antes de que se produzcan los daños nos puede ser de gran utilidad.

Los sistemas Linux nos permiten hacer una copia de seguridad del *MBR* haciendo uso del comando *dd*. Por ejemplo:

```
$> dd if=/dev/hda of=~/.MBR.backup bs=512 count=1
```

Donde */dev/hda* es una referencia al primer disco duro *IDE* (alternativamente se puede utilizar */dev/sda* para hacer referencia al primer disco duro *SCSI*) y *~/.MBR.backup* indica que el archivo *MBR.backup* contendrá la copia de seguridad de nuestro *MBR* el cual será almacenado en nuestro directorio de conexión. Otra opción que se debe especificar es el tamaño del bloque a través de *bs*, el cual será de 512 bytes. Y por último establecemos la cuenta (*count*) a sólo uno, ya que sólo queremos un sector.

Para restaurar un registro de inicio maestro, lo único que debemos hacer es invertir los parámetros *if* y *of* del comando *dd*.

```
$> dd if=~/.MBR.backup of=/dev/hda bs=512 count=1 conv=notrunc
```

Si estuviésemos interesados en borrar el registro de inicio maestro del disco duro principal, por cualquier circunstancia (por ejemplo, porque necesitamos eliminar la información de este sector), lo único que debemos hacer es escribir el siguiente comando:

```
$> dd if=/dev/zero of=/dev/hda bs=512 count=1
```


X. PRÁCTICAS DE LABORATORIO

Administración de sistemas operativos

Práctica 0: Programación de *shell scripts*

OBJETIVOS

- Estudiar los principales aspectos del lenguaje *bash* para programar *shell scripts*.
- Entender la filosofía de programación de *shell scripts* mediante ejemplos sencillos.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de dos sesiones de laboratorio, cada una de dos horas para un total de cuatro horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

COMPLEMENTARIA

El shell Bash

Dirección: <http://personales.ya.com/macprog/bash.pdf>

Programación en BASH – COMO de introducción

Dirección: <http://www.ibiblio.org/pub/Linux/docs/HOWTO/translations/es/pdf/Bash-Prog-Intro-COMO.pdf>

PRÁCTICA 0
Programación de *shell scripts*

TABLA DE CONTENIDOS:

Introducción.....	241
Primer programa de <i>shell</i>	242
Sustituciones de órdenes por su salida.....	243
Mecanismos de escape.....	243
Barra invertida.....	243
Comillas simples.....	243
Comillas dobles.....	244
Paso de parámetros a un programa de <i>shell</i>	244
Algunas variables especiales de la <i>shell</i>	244
Construcciones del lenguaje.....	245
shift.....	246
read.....	247
expr.....	248
Operadores aritméticos.....	248
Operadores relacionales.....	249
Operadores lógicos.....	250
Evaluaciones.....	251
test (para archivos).....	251
test (para evaluación de cadenas).....	252
test (para evaluaciones numéricas).....	252
if.....	253
case.....	255
while.....	256
until.....	257
for.....	257
break, continue y exit.....	259
select.....	259
Uso de funciones en programas de <i>shell</i>	260
Ejercicios.....	262

Introducción

El *shell* es un intérprete de órdenes, pero el *shell* no es solamente eso; los intérpretes de órdenes de Unix son auténticos lenguajes de programación. Como tales, incorporan sentencias de control de flujo, sentencias de asignación, funciones, etc. Los programas de *shell* no necesitan ser compilados como ocurre en otros lenguajes. En este caso es el propio *shell* el que se encarga de interpretarlos línea a línea. A estos programas se les conoce generalmente como *shell scripts*, y son los equivalentes a los archivos por lotes de otros sistemas operativos.

Cabe destacar que a pesar de que en los sistemas Unix no existen las extensiones de los archivos (*.txt*, *.exe*, etc.) es recomendable asignarle a los archivos de *script* la extensión *.sh*, para que de esta manera sepamos que corresponde a un *script* particular escrito por nosotros.

Esta práctica esta orientada al aprendizaje de la programación de *shell script* mediante el lenguaje *bash* (*bourne-again shell*) el cual es un lenguaje interpretado de programación que ayuda al administrador de sistemas a realizar la mayor parte de las tareas necesarias, tanto en la automatización como en el arranque del sistema.

En la presente práctica se ampliarán aspectos del lenguaje *bash* mostrando su aplicación para la programación de *shell scripts*. Sin embargo, la mayor parte de las órdenes, definiciones de variables, constantes, etc. pueden utilizarse directamente por la línea de órdenes.

Primer programa de *shell*

Vamos a crear a continuación un sencillo *shell script* para mostrar cual va a ser la técnica general para crear este tipo de programas. En primer lugar, lo que tenemos que hacer es elegir el nombre que le vamos a dar a nuestro programa. En nuestro caso debemos ser originales y asignarles nombre en función de lo se pretenda que va a ejecutar el *script*. A continuación invocaremos a nuestro editor favorito (*vim*, *emacs*, *gedit*, *kate*, etc.) e introduciremos dos líneas de texto correspondientes a dos órdenes Unix. Con ello generaremos un archivo que contiene lo siguiente:

```
$ cat script_prueba.sh
#!/bin/bash
# Shell script de prueba
who # Mostramos a los usuarios que están actualmente conectados en el sistema
date # Imprimimos la fecha y hora del sistema
$
```

Debido a que existen diferentes tipos de *shell* (*csh*, *sh*, *tcsh*, *zsh*, *bash*, etc.), cada una con un conjunto de órdenes internas, es necesario que para el correcto funcionamiento de nuestros *shell scripts*, tengamos que indicar al comienzo del *script* cuál va a ser la *shell* que lo va a interpretar. El modo de conseguirlo, es colocando una primera línea en el *script* que comience con los caracteres *#!* seguido de la ruta hasta el ejecutable de la *shell* a utilizar. Para conocer cual es la *shell* implementada en nuestro sistema podemos hacer uso del siguiente comando: **echo \$SHELL** (el resultado debe ser */bin/bash*), y lo que aquí aparezca es lo que hemos de colocar después de los caracteres *#!*.

NOTA: En esta práctica sólo se describirá la sintaxis reconocida por *bash*. En caso de encontrar otra *shell* implementada en su sistema que sea diferente de *bash* favor de no continuar con la práctica e indicarle al docente dicho problema o probar establecer el *shell bash* como predeterminado mediante el uso del siguiente comando: **chsh -s /bin/bash**.

Una vez creado el archivo de texto (*script_prueba.sh*), debemos cambiar sus atributos para que tenga derecho de ejecución. La forma de hacerlo es la siguiente:

```
$ chmod +x script_prueba.sh
```

Una vez cambiados los derechos, ya podemos ejecutar nuestro programa de la siguiente forma: **./script_prueba.sh**. Sólo es necesario poner al archivo el atributo de ejecución una vez, puesto que una vez cambiado, este atributo no se verá modificado. Así pues, aunque volvamos a editar el archivo, no será necesario utilizar de nuevo la orden *chmod*. Los resultados de la ejecución del programa se muestran seguidamente:

```
$ ./script_prueba.sh
gateway :0      2008-05-27 00:37
mar may 27 05:45:24 CST 2008
$
```

Es posible (e incluso recomendable), tal y como hemos hecho en este primer ejemplo, añadir comentarios a nuestros programas de *shell*, para ello, si una línea es de comentarios, debemos comenzar con el carácter *#*.

Sustituciones de órdenes por su salida

Esta técnica consiste en sustituir el nombre de una orden por su salida estándar. Para realizar esta transformación, la orden ha de introducirse entre los caracteres especiales `...` (tildes graves). Esta notación se conoce también como antigua sustitución de órdenes.

Empleando esta notación se eliminan los caracteres de escape (\) que preceden a los caracteres especiales \$ y \ antes de ejecutar la orden. Esto permite incluir estos caracteres evitando que sean interpretados por la *shell*. Ejemplo:

```
$ echo El directorio actual de trabajo es: `pwd`  
El directorio actual de trabajo es: /home/luis  
$
```

Mecanismos de escape

Los mecanismos de escape son aquellos que sirven para evitar que la *shell* interprete caracteres especiales o palabras reservadas con un significado habitual, tomándolos entonces como literales. Estos mecanismos evitan, por tanto, la expansión de parámetros o la sustitución de órdenes.

Los caracteres empleados como mecanismos de escape son la barra invertida (\), las comillas simples ('...') y las comillas dobles ("...").

Barra invertida

La barra invertida tiene las siguientes interpretaciones dependiendo de la situación en la que se encuentre: Si no aparece encerrado entre comillas (simples o dobles), se elimina y el carácter al que precede es tomado como literal. Si el carácter al que acompaña es el de la línea nueva, *newline*, también se elimina este carácter. Este comportamiento se denomina continuación de línea y permite escribir texto en varias líneas y que la *shell* las considere como una sola.

Si forma parte del texto encerrado entre comillas simples mantiene su significado literal (no es interpretada). Cuando se encuentra dentro de un texto encerrado entre comillas dobles mantiene su significado literal, excepto cuando se usa para escapar los caracteres \$, ' y ". Ejemplo:

```
$ echo \$HOME es $HOME  
$HOME es /home/luis  
$
```

Comillas simples

Los caracteres situados dentro de las comillas simples pierden su significado y son tomados como literales. La comilla simple no puede aparecer dentro del literal puesto que delimitaría el final de la cadena. Ejemplo:

```
$ echo '$HOME, ", \n, ls'  
HOME, ", \n, ls  
$ echo '\ 'Esto se encuentra dentro de: ('...'') comillas simples' \  
'Esto se encuentra dentro de: ('...') comillas simples'  
$
```

Comillas dobles

Eliminan la interpretación de todos los caracteres que encierran excepto: \$, ', " y \. Ejemplo:

```
$ echo "\$HOME es $HOME"
$HOME es /home/luis
$
```

Paso de parámetros a un programa de shell

A menudo queremos que nuestros programas de *shell* reciban parámetros desde la línea de órdenes para hacerlos más versátiles. Estos parámetros son los que se conocen como parámetros de posición. Los parámetros de posición se pueden usar dentro de un programa del *shell* como cualquier otra variable del *shell*; es decir, para saber su valor utilizaremos el símbolo \$. Los parámetros dentro del *shell script* son accesibles utilizando

- **\$0** Representa al parámetro cero o nombre del programa.
- **\$1** Representa al parámetro uno.
- ...
- **\$9** Representa al parámetro nueve. Aunque una línea de órdenes puede tener más de nueve parámetros, un programa *shell* sólo puede acceder directamente a los primeros nueve parámetros mediante **\$1**, **\$2**, ..., **\$9**; para parámetros superiores a nueve se emplea la notación **\${numero_parametro}** o se emplea la orden *shift* como se mostrará más adelante.

Sí, por ejemplo, tenemos un programa de *shell* denominado *prog.sh* y lo invocamos de la siguiente forma:

```
$ ./prog.sh datos 35 suma
```

Dentro del programa de *shell* tenemos lo siguiente:

```
$0 = ./prog.sh
$1 = datos
$2 = 35
$3 = suma
```

Vamos a poner un ejemplo de un *shell script* que visualiza los cuatro primeros parámetros que le pasemos. Al programa lo denominaremos *param.sh*, y su contenido es el siguiente:

```
$ cat param.sh
#!/bin/bash
# Este script visualiza los parámetros que le pasamos desde la línea de órdenes
echo Parámetro 0 = $0
echo Parámetro 1 = $1
echo Parámetro 2 = $2
echo Parámetro 3 = $3
$
```

Algunas variables especiales de la shell

Dentro de un programa de *shell* existen variables con significados especiales, algunas de las cuales se citan a continuación:

- `$#` Esta variable guarda el número de argumentos de la línea de órdenes (excluyendo el nombre del programa).
- `$*` Esta variable guarda la cadena de argumentos entera (excluyendo el nombre del programa).
- `$?` Esta variable guarda el código de retorno de la última orden ejecutada (0 si no hay error y distinto de 0 si hay error).
- `@$` Esta variable representa la cadena de argumentos entera (excluyendo el nombre del programa) pero como una lista de cadenas, a diferencia de `$*` que obtiene todos los argumentos como una única cadena.

Vamos a mostrar con un sencillo ejemplo el uso de estas variables. En este caso, el nombre del *shell script* será *var.sh*

```
$ cat var.sh  
#!/bin/bash  
# Programa shell que visualiza las variables #, * y ?  
echo La variable \# vale: $#  
echo La variable \* vale: $*  
cp          # Forzamos un error ya que al comando cp le faltan argumentos  
echo La variable \? vale: $?  
$
```

Como podemos apreciar, cualquier carácter susceptible de ser interpretado por la *shell* es precedido por el carácter *backslash* (`\`) para que pierda su significado especial. Ahora para probar el programa lo lanzamos con una serie de argumentos:

```
$ ./var.sh uno dos tres cuatro  
La variable # vale: 4  
La variable * vale: uno dos tres cuatro  
cp: falta un archivo como argumento  
Pruebe 'cp -help' para más información.  
La variable ? vale: 1  
$
```

Como podemos observar, la variable `$?` toma un valor distinto de cero, puesto que la orden *cp* se ha ejecutado con errores. Es importante que si dentro de un programa de *shell*, se produce algún error tomemos decisiones al respecto. Como veremos más adelante, existen mecanismos para tomar diferentes caminos en función del resultado de la ejecución de una orden.

Construcciones del lenguaje

Vamos a ver seguidamente las construcciones del lenguaje típicas empleadas en los programas de *shell*. No vamos a realizar una descripción exhaustiva de todas y cada una de las construcciones, sino que nos vamos a centrar en lo empleado más comúnmente.

shift

Sintaxis: `shift n`

Esta orden se utiliza para desplazar los argumentos, de manera que `$2` pasa a ser `$1`, `$3` pasa a ser `$2`, y así sucesivamente (esto si el desplazamiento `n` es igual a 1). Es muy utilizado dentro de los bucles. Vamos a poner un ejemplo con un programa que denominaremos `shift1.sh`, cuyo contenido se muestra a continuación:

```
$ cat shift1.sh
#!/bin/bash
# Programa de shell que muestra el uso de shift
echo \"$1 vale: $1
echo \"$2 vale: $2
echo \"$3 vale: $3
shift 2
echo Ahora \"$1 vale: $1
echo Ahora \"$2 vale: $2
echo Ahora \"$3 vale: $3
$
```

En el ejemplo anterior, al desplazar dos lugares tendremos que `$5` pasa a ser `$3`, `$4` pasa a ser `$2` y `$3` pasa a ser `$1`. Los argumentos iniciales, `$1` y `$2`, se pierden después del desplazamiento. Vamos a ejecutar el programa anterior:

```
$ ./shift1.sh uno dos tres cuatro cinco
$1 vale: uno
$2 vale: dos
$3 vale: tres
Ahora $1 vale: tres
Ahora $2 vale: cuatro
Ahora $3 vale: cinco
$
```

Evidentemente este desplazamiento afecta también a las variables `#` y `*`. Veamos otro ejemplo, que denominaremos `shif2.sh`

```
$ cat shift2.sh
#!/bin/bash
# Otro ejemplo con shift
echo \"$# vale: $#
echo \"$* vale: $*
shift 2
echo Ahora \"$# vale: $#
echo Ahora \"$* vale: $*
$
```

Al ejecutar el programa anterior, se produce el siguiente resultado:

```
$ ./shift2 uno dos tres cuatro cinco
$# vale: 5
$* vale: uno dos tres cuatro cinco
Ahora $# vale: 3
Ahora $* vale: tres cuatro cinco
$
```

La orden *shift* desplaza todas las cadenas en * a la izquierda *n* posiciones y decrementa # en *n*. Si a *shift* no se le indica el valor de *n*, por defecto tomará el valor 1. La orden *shift* no afecta al parámetro de posición 0 o nombre del programa.

read

Sintaxis: `read variable(s)`

La orden *read* se usa para leer información escrita en el terminal de forma interactiva. Si hay más variables en la orden *read* que palabras escritas, las variables que sobran por la derecha se asignarán a *NULL*. Si se introducen más palabras que variables haya, todos los datos que sobran por la derecha se asignarán a la última variable de la lista.

En el siguiente ejemplo, el programa *read1.sh* va a leer una variable desde la entrada estándar, y posteriormente va a visualizar esa variable por la salida estándar.

Ejemplo:

```
$ cat read1.sh
#!/bin/bash
# Programa que ilustra el uso de la orden read
# La opción -n en la orden echo, se emplea para evitar el retorno de carro
echo -n Introduce una variable:
read var
echo La variable introducida es: $var
$
```

Cuando ejecutemos este programa, obtendremos el siguiente resultado:

```
$ ./read1.sh
Introduce una variable: 123
La variable introducida es: 123
$
```

A continuación analizaremos el caso en que leemos más o menos variables de las que queremos leer desde el programa *shell*. Para ello, consideremos el siguiente programa, que lee tres variables. En un primer caso vamos a introducir sólo dos, y en un segundo introduciremos más de tres variables. El código del programa en cuestión es el siguiente:

```
$ cat read2.sh
#!/bin/bash
# Programa que lee varias variables con read
echo -n Introduce las variables:
read var1 var2 var3
echo Las variables introducidas son:
echo var1 = $var1
echo var2 = $var2
echo var3 = $var3
$
```

Veamos una ejecución normal en la que leemos tres variables:

```
$ ./read2.sh  
Introduce las variables: 34 hola 938  
Las variables introducidas son:  
var1 = 34  
var2 = hola  
var3 = 938  
$
```

Vamos a ejecutar el programa anterior introduciendo sólo dos variables:

```
$ ./read2.sh  
Introduce las variables: uno dos  
Las variables introducidas son:  
var1 = uno  
var2 = dos  
var3 =  
$
```

Como podemos observar, la variable *var3* queda sin asignar, puesto que sólo hemos introducido dos valores. A continuación ejecutaremos de nuevo el programa, pero ahora introduciremos cuatro variables:

```
$ ./read2.sh  
Introduce las variables: uno dos tres cuatro  
Las variables introducidas son:  
var1 = uno  
var2 = dos  
var3 = tres cuatro  
$
```

En este caso a la variable *var3* se le asignan todas las variables a partir de la dos.

expr

Sintaxis: `expr arg1 op arg2 [op arg3 ...]`

Los argumentos de la orden *expr* se toman como expresiones y deben ir separados por espacios en blanco. La orden *expr* evalúa sus argumentos y escribe el resultado en la salida estándar. El uso más común de la orden *expr* es para efectuar operaciones de aritmética simple y, en menor medida, para manipular cadenas (averiguar la longitud de una cadena, filtrar determinados caracteres de una cadena, etc.).

Operadores aritméticos

Los siguientes operadores se utilizan para evaluar operaciones matemáticas y escribir el resultado de la operación por la salida estándar. Las operaciones que podemos realizar son las siguientes: suma, resta, multiplicación, división entera y cálculo del resto de la división entera.

- + Suma *arg2* a *arg1*.
- - Resta *arg2* a *arg1*.
- * Multiplica los argumentos.
- / Divide *arg1* entre *arg2* (división entera).
- % Resto de la división entera entre *arg1* y *arg2*.

En el caso de utilizar varios operadores, las operaciones de suma y resta se evalúan en último lugar, a no ser que vayan entre paréntesis. No hay que olvidar que los símbolos `*`, `(` y `)` tienen un significado especial para la *shell*, por lo deben ser precedidos por el símbolo *backslash* o encerrados entre comillas simples.

Ejemplo:

```
$ cat expr1.sh
#!/bin/bash
# Programa de shell que multiplica dos variables leídas desde el teclado
echo
echo Multiplicación de dos variables
echo -----
echo
echo -n Introduce la primera variable:
read arg1
echo -n Introduce la segunda variable:
read arg2
resultado=`expr $arg1 \* $arg2`
echo Resultado=$resultado
$
```

El resultado de ejecutar el programa anterior es el producto de las dos variables leídas desde el teclado.

Veamos un uso particular:

```
$ ./expr1.sh

Multiplicación de dos variables
-----

Introduce la primera variable: 12
Introduce la segunda variable: 20
Resultado=240
$
```

Operadores relacionales

Estos operadores se utilizan para comparar dos argumentos. Los argumentos pueden ser también palabras. Si el resultado de la comparación es cierto, el resultado es uno; si es falso, el resultado es cero. Estos operadores se utilizan mucho para comparar operandos y tomar decisiones en función de los resultados de la comparación. Veamos los distintos tipos de operadores relacionales:

- `=` ¿Son los argumentos iguales?
- `!=` ¿Son los argumentos distintos?
- `>` ¿Es *arg1* mayor que *arg2*?
- `>=` ¿Es *arg1* mayor o igual que *arg2*?
- `<` ¿Es *arg1* menor que *arg2*?
- `<=` ¿Es *arg1* menor o igual que *arg2*?

No olvide que los símbolos `>` y `<` tiene significado especial para la *shell*, por lo que deben ser entrecorridos.

Ejemplo:

```
$ cat expr2.sh
#!/bin/bash
# Programa de shell que determina si dos variables leídas desde el
# teclado son iguales o no
echo
echo Son iguales las variables?
echo -----
echo
echo -n Introduce la primera variable:
read arg1
echo -n Introduce la segunda variable:
read arg2
resultado=`expr $arg1 = $arg2`
echo Resultado=$resultado
$
```

El programa anterior devolverá cero si las dos variables introducidas son distintas y uno si son iguales.

Operadores lógicos

Estos operadores se utilizan para comparar dos argumentos. Dependiendo de los valores, el resultado puede ser *arg1* (o alguna parte de él), *arg2* o cero. Como operadores lógicos tenemos los siguientes:

- **/ OR lógico.** Si el valor de *arg1* es distinto de cero, el resultado es *arg1*; si no es así, el resultado es *arg2*.
- **& AND lógico.** Si *arg1* y *arg2* son distintos de cero, el resultado es *arg1*; si no es así, el resultado es *arg2*.
- **:** El *arg2* es el patrón buscado en *arg1*. Si el patrón *arg2* está encerrado dentro de paréntesis $\backslash()$, el resultado es la parte de *arg1* que coincide con *arg2*. Si no es así, el resultado es simplemente el número de caracteres que coinciden.

No olvide que los símbolos / y & deben ser entrecomillados o precedidos del símbolo \, por tener un significado especial para la *shell*. Veamos ahora algunos ejemplos en los que invocamos a *expr* desde la línea de órdenes:

```
$ a=5
$ a=`expr $a + 1`
$ echo $a
6
$
```

En este primer ejemplo hemos incrementado en una unidad el valor de la variable *a*.

```
$ a=palabra
$ b=`expr $a : .*`
$ echo $b
7
$
```

En este ejemplo hemos calculado el número de caracteres de la cadena *a*.

```
$ a=junio_2004
$ b=`expr $a : '\([a-z]*\)´
$ echo $b
junio
$
```

En este último ejemplo hemos determinado cuáles son los caracteres comprendidos entre la letra *a* y la *z* minúsculas en la cadena *a*.

Evaluaciones

Sirven para averiguar el valor lógico de una determinada expresión. Habitualmente su uso se combina con una instrucción de bifurcación, como por ejemplo *if*.

test (para archivos)

Sintaxis: `test` *-opción* argumento [*-opción* argumento]

La orden *test* se usa para evaluar expresiones y generar un valor de retorno; este valor no se escribe en la salida estándar, pero asigna cero al código de retorno si la expresión se evalúa como verdadera, y le asigna uno si la expresión se evalúa como falsa. Se puede invocar la orden *test* también mediante [expresión], tanto a la derecha como a la izquierda de expresión debe haber un espacio en blanco. *test* puede evaluar tres tipos de elementos: archivos, cadenas y números.

Opciones:

- **-f** Devuelve verdadero (0) si el archivo existe y es un archivo regular (no es un directorio ni un archivo de dispositivo).
- **-s** Devuelve verdadero (0) si el archivo existe y tiene un tamaño mayor que cero.
- **-r** Devuelve verdadero (0) si el archivo existe y tiene permiso de lectura.
- **-w** Devuelve verdadero (0) si el archivo existe y tiene permiso de escritura.
- **-x** Devuelve verdadero (0) si el archivo existe y tiene permiso de ejecución.
- **-d** Devuelve verdadero (0) si el archivo existe y es un directorio.

Ejemplo:

```
$ test -f archivo32
$ echo $?
1
$ test -f /etc/passwd
$ echo $?
0
$
```

test (para evaluación de cadenas)

Sintaxis: test cadena1 operador cadena2
[cadena1 operador cadena2]

Ejemplo:

```
$ a=palabra1
$ [ $a = palabra2 ]
$ echo $?
1
$ [ $a = palabra1 ]
$ echo $?
0
$
```

De esta manera, *test* evalúa si las cadenas son iguales o distintas. Cuando se evalué una variable del *shell*, es posible que dicha variable no contenga nada. Consideremos el siguiente caso:

```
[ $var = vt100 ]
```

Si a *var* no le hemos asignado nada, el *shell* realizará la sustitución de variables, y la orden que el *shell* intentará ejecutar será la siguiente:

```
[ = vt100 ]
```

Lo cual nos dará un error de sintaxis. Una forma sencilla de evitarlo consiste en meter entre comillas la variable que vamos a evaluar, y así sabremos que la variable tomará el valor *NULL*.

```
[ "$var" = vt100 ]
```

Si como en el ejemplo anterior, *\$var* no contiene ningún valor, la expresión que verá *test*, una vez procesada por el *shell* será:

```
[ "" = vt100 ]
```

Esta expresión es sintácticamente correcta y no provocará ningún error de sintaxis.

test (para evaluaciones numéricas)

Sintaxis: test numero1 operador numero2
[numero1 operador numero2]

En evaluaciones numéricas esta orden es sólo válida con números enteros. Los operadores usados para comparar números son diferentes de los usados para comparar cadenas. Estos operadores numéricos son:

- **-lt** Menor que.
- **-le** Menor o igual que.
- **-gt** Mayor que.
- **-ge** Mayor o igual que.
- **-eq** Igual a.
- **-ne** No igual a.

Hay unos cuantos operadores que son válidos en una expresión de la orden *test* a la hora de evaluar tanto archivos como cadenas o números. Estos operadores son:

- **-o** OR
- **-a** AND
- **!** NOT

Ejemplo:

```
$ a=23
$ [ $a -lt 55 ]
$ echo $?
0
$ test $a != 23
$ echo $?
1
$
```

if

```
Sintaxis: if condicion1
      then
          orden1
      [elif condicion2
      then
          orden2]
      ...
      [else
          orden3]
      fi
```

La construcción *if* se utiliza para tomar decisiones a partir de los códigos de retorno, normalmente devueltos por la orden *test*. La ejecución de la instrucción *if* es tal como sigue:

1. Se evalúa la *condicion1*.
2. Si el valor de retorno de *condicion1* es verdadero (0), se ejecutará *orden1*.
3. Si esto no es así y se cumple la *condicion2*, se ejecutará la *orden2*.
4. En cualquier otro caso, se ejecuta *orden3*.

Ejemplo:

```
$ cat if.sh
#!/bin/bash
# Shell script que muestra el uso de la sentencia de control if-fi
if test -f /etc/hosts
then
    cat /etc/hosts
else
    echo El archivo no existe
fi
$
```

En el ejemplo anterior, si existe el archivo */etc/hosts*, entonces lo visualizaremos. Si no existe, imprimiremos por pantalla un mensaje diciendo que tal archivo no existe.

A continuación vamos a poner otro ejemplo, en el cual, si no existe un directorio, lo crearemos desde un programa de *shell* y le habilitaremos los derechos de modo que sólo el propietario tenga acceso a él. El nombre del archivo se le pasa como parámetro al *shell script*. El contenido del programa es el siguiente:

```
$ cat crea.sh
#!/bin/bash
# Ejemplo de uso de if. Este programa crea si (no existe) el archivo que le
# indiquemos desde la línea de órdenes. Al directorio recién creado sólo
# tendrá acceso el propietario del mismo.
if [ !-d $1 ]
then
    mkdir $1
    chmod 700 $1
fi
$
```

En el siguiente ejemplo vamos a diseñar un *shell script* que admita un argumento. Si el argumento dado coincide con el nombre de un archivo o directorio, deberá sacar por pantalla de qué tipo es. Si es además un archivo, deberá determinar si es ejecutable o no.

```
$ cat if2.sh
#!/bin/bash
# Programa shell que comprueba si existe un archivo pasado como argumento
# y si existe muestra de que tipo es.
if [ $# = 0 ]
then
    echo Debes introducir al menos un argumento.
    exit 1
fi
if [ -f "$1" ]
then
    # Es un archivo regular
    echo -n "$1 es un archivo regular "
    if [ -x "$1" ]
    then
        echo "ejecutable."
    else
        echo "no ejecutable."
    fi
elif [ -d "$1" ]
then
    # Es un directorio
    echo "$1 es un directorio"
else
    # Es una cosa rara
    echo "$1 es una cosa rara o no existe"
fi
$
```

if también puede utilizarse para comprobar el resultado de la ejecución de un programa externo, ya que todos los programas en Unix devuelven un valor numérico como resultado de su ejecución, que indica si dicha ejecución se llevo a cabo correctamente o no.

Por ejemplo, podemos diseñar un *shell script* que compruebe si existe un determinado usuario en el archivo de contraseñas. Para ello vamos a utilizar una expresión regular interpretada por *grep*. El programa de *shell* podría ser el siguiente:

```
$ cat comp_usuario.sh
#!/bin/bash
# Programa shell que comprueba la existencia de un usuario en el
# archivo de contraseñas.
if grep -q '^$1:' /etc/passwd
then
    echo El usuario $1 ya existe en el sistema.
else
    echo El usuario $1 no existe en el sistema.
fi
$
```

Podemos ampliar el programa anterior para averiguar si el usuario, de existir, es un usuario regular (su UID es mayor que 1000).

```
$ cat comp_usuario2.sh
#!/bin/bash
# Programa shell que comprueba la existencia de un usuario en el archivo
# de contraseñas y verifica si se trata de un usuario regular
if grep -q '^$1:' /etc/passwd
then
    echo El usuario $1 ya existe en el sistema.
    UID=`cat /etc/passwd | grep '^$1:' | cut -f3 -d':'`
    if [ UID -ge 1000 ]
    then
        echo $1 es un usuario regular.
    else
        echo $1 no es un usuario regular.
    fi
else
    echo El usuario $1 no existe en el sistema.
fi
$
```

case

Sintaxis: case palabra in
 patron1)
 orden1
 ;;
 patron2)
 orden2
 ;;
 ...
 patronN)
 ordenN
 ;;
 esac

La construcción *case* controla el flujo del programa basándose en la palabra dada. La palabra se compara, en *orden*, con todas las platillas. Cuando se encuentra la primera que corresponde, se ejecuta la lista de órdenes asociadas, la cual tiene que terminar con dos punto y coma (;).

Ejemplo:

```
$ cat case.sh
#!/bin/bash
# Programa que ilustra el uso de la sentencia de control de flujo case-esac.
dia=`date | cut -c 0-3`
case dia in
    lun)
        echo Hoy es Lunes
        ;;
    mar)
        echo Hoy es Martes
        ;;
    mie)
        echo Hoy es Miércoles
        ;;
    jue)
        echo Hoy es Jueves
        ;;
    vie)
        echo Hoy es Viernes
        ;;
    sab)
        echo Hoy es Sabado
        ;;
    dom)
        echo Hoy es Domingo
        ;;
esac
$
```

El programa anterior puede ser utilizado para saber el día de la semana, visualizando los resultados en español. Obsérvese cómo en la variable *dia* almacenamos lo que retorna la orden *date | cut -c 0-3*, que son las tres primeras letras del día de la semana.

while

Sintaxis: while condición
do
orden(es)
done

La ejecución de la construcción *while* es como sigue:

1. Se evalúa la condición.
2. Si el código devuelto por la condición es verdadero (0), se ejecutará la orden u órdenes y se vuelve a iterar.
3. Si el código de retorno de la condición es falso (1), se saltará a la primera orden que haya después de la palabra reservada *done*.

Ejemplo:

```
$ cat while.sh
#!/bin/bash
# Programa que ilustra el uso de la sentencia de control de flujo while.
a=42
while [ $a -le 53 ]
do
    echo contador = $a
    a=`expr $a + 1`
done
$
```

En el anterior ejemplo se incrementa y visualiza el valor del contador mientras éste sea menor o igual que 53. Para ello, *while* comprueba el código de retorno de la orden `[$a -le 53]`, y si es cierto, se repite la iteración.

until

Sintaxis: until condición

```
do
    orden(es)
done
```

La construcción *until* es muy similar a la de *while*. La ejecución es como sigue:

1. Se evalúa la condición.
2. Si el código de retorno de la condición es distinto de 0 (falso), se ejecutará la orden u órdenes y se vuelve a iterar.
3. Si el código devuelto por la condición es 0 (verdadero), se saltará a la primera orden que haya después de la palabra clave *done*.

Ejemplo:

```
$ cat until.sh
#!/bin/bash
# Programa que ilustra el uso de la sentencia de control de flujo until.
until [ $a = hola ]
do
    echo -n Introduce una cadena:
    read a
done
$
```

En el ejemplo anterior, el bucle *until* se ejecuta hasta que el usuario introduzca la cadena *hola*. A partir de este momento, la condición se vuelve verdadera y se termina el bucle.

for

Sintaxis: for variable in lista

```
do
    orden(es)
done
```

variable puede ser cualquier variable del *shell*, y *lista* es una lista compuesta de cadenas separadas por espacios en blanco o tabuladores. La construcción funciona como sigue:

1. Se asigna a *variable* la primera cadena de la *lista*.
2. Se ejecuta orden.
3. Se asigna a *variable* la siguiente cadena de la *lista*. Se vuelve a ejecutar orden.
4. Repetir hasta que se hayan usado todas las cadenas.
5. Después de que haya acabado el bucle, la ejecución continua en la primera línea que sigue a la palabra clave *done*.

Ejemplo:

```
$ cat for.sh
#!/bin/bash
# Programa que ilustra el uso de la sentencia de control de flujo for.
for i in manuel ana carlos miguel
do
    mail $i < carta
done
$
```

En el ejemplo anterior se envía el archivo *carta* a todos los usuarios indicados en la lista. Si dentro del bucle *for* omitimos lista, se asumirá como lista el parámetro de posición *\$@* que representa la cadena de argumentos entera excluyendo el nombre del programa.

El siguiente ejemplo es una modificación del *shell script* (*if2.sh*) visto en la parte correspondiente a *if* para que pueda tratar con varios archivos pasados como argumento. El código del programa es el siguiente:

```
$ cat modifif2.sh
#!/bin/bash
# Programa shell que comprueba si existe un archivo pasado como argumento y si
# existe muestra de qué tipo es
if [ $# = 0 ]
then
    echo Debes introducir al menos un argumento.
    exit 1
fi
for i in $@
do
    if [ -f "$1" ]
    then
        # Es un archivo regular
        echo -n "$1 es un archivo regular "
        if [ -x "$1" ]
        then
            echo "ejecutable."
        else
            echo "no ejecutable."
        fi
    elif [ -d "$1" ]
    then
        # Es un directorio
        echo "$1 es un directorio"
    else
```

```

        # Es una cosa rara
        echo "$1 es una cosa rara o no existe"
    fi
    # Ahora desplazamos los argumentos
    shift
done
$

```

break, continue y exit

- **break [n]** Hace que cualquier bucle *for*, *while* o *until* termine y pase el control a la siguiente orden que se encuentre después de la palabra reservada *done*.
- **continue [n]** Detiene la iteración actual del bucle *for*, *while* o *until* y empieza la ejecución de la siguiente iteración.
- **exit [n]** Detiene la ejecución del programa del *shell* y asigna *n* al código de retorno (normalmente cero implica éxito, y distinto de cero, error).

Ejemplo:

```

$ cat exit.sh
#!/bin/bash
# Programa shell que te saca si el número de parámetros es igual a cero
if [ $# -eq 0 ]
then
    echo Forma de uso: $0 [-c] [-d] archivo(s)
    exit 1 # Código de retorno erróneo
fi
$

```

La secuencia de código anterior puede ser utilizada dentro de un programa de *shell* para comprobar si le pasamos o no parámetros. En caso de no pasarle parámetros, visualizará el mensaje de error y terminará el programa.

select

Sintaxis: `select i [in lista]`
 do
 orden(es)
done

Esta sentencia visualiza los elementos indicados en lista, numerados en el orden en que aparecen, en la salida estándar de error. Si no se proporciona tal lista, ésta es leída desde la línea de órdenes a través de la variable `$@`. A continuación de las opciones numeradas indicadas en lista se visualiza la cadena (*prompt*), indicada por la variable `PS3`. Cuando aparezca este *prompt*, tendremos que elegir una de las opciones indicadas en la lista introduciendo el número que la identifica. Si se introduce una opción válida, se ejecutarán las órdenes asociadas. Si como opción introducimos *Enter*, el menú de opciones volverá a ser visualizado. Cualquier entrada que indique el usuario será almacenada en el variable `REPLY`.

Ejemplo:

```
$ cat select.sh
#!/bin/bash
# Programa que ilustra el uso de la sentencia select
PS3="Opcion: "
select i Listado Quien Salir
do
    case $i in
        Listado)
            ls -l
            ;;
        Quien)
            who
            ;;
        Salir)
            exit 0
            ;;
        *)
            echo Opcion incorrecta
    esac
done
$
```

Uso de funciones en programas de *shell*

Dentro de los programas de *shell* se puede hacer uso de funciones. En una función podemos agrupar un conjunto de órdenes que se ejecuten con cierta frecuencia. Las funciones hay que declararlas antes de usarlas.

Ejemplo:

```
$ cat funciones.sh
#!/bin/bash
# Si no se pasan parámetros al programa se ejecuta la función error. Obsérvese
# que para invocar a la función no colocamos los paréntesis. Seguidamente
# definimos la función error.
error()
{
    echo Error de sintaxis
    exit 1
}
if [ $# = 0 ]
then
    error
else
    echo Hay $# argumentos
fi
$
```

Las funciones además pueden colocarse en otro archivo aparte. De esta forma podemos diseñar una biblioteca de funciones y reutilizarlas en nuestros programas.

Como ejemplo de aplicación de funciones vamos a diseñar una función que denominaremos *espacio_ocupado(id_particion)* que obtenga la cantidad de espacio ocupado de una partición de

disco dada como argumento. Esta función la vamos a situar en un archivo aparte denominado *funciones.sh*.

Para diseñar la función partiremos de la información que nos aporta la orden *df* cuya salida es similar a la siguiente:

```
$ df
S. ficheros    1K-blocks    Used    Available    Use %    Montado en
/dev/hda2      7384424     6090076    919232     87%     /
none          119624      0         119624     0%     /dev/shm
$
```

Esta orden nos informa de que la partición *hda2* tiene 6090076 bytes ocupados. Podemos utilizar el filtro *cut* para obtener sólo este campo y *grep* para localizar la línea que contiene la información sobre la partición en la que estemos interesados:

```
$ df -k | grep /dev/hda2 | tr -s ' ' | cut -f3 -d ' '
6090076
$
```

Utilizamos el modificador *-k* para que el resultado de *df* esté expresado en kilobytes. La orden *tr -s* suprime los espacios en blanco duplicados para que *cut* pueda usarlos como delimitador de campos de forma correcta.

Ahora que tenemos la orden correcta vamos a introducirla en el archivo *funciones.sh*:

```
$ cat funciones.sh
#!/bin/bash
espacio_ocupado()
{
    ESPACIO=`df -k | grep /dev/$particion | tr -s ' ' | cut -f3 -d ' '`
}
$
```

Para hacer uso de esta función desde otro *script* es necesario indicar en que archivo se encuentra. Para esto se coloca al principio de la línea un punto, un espacio y el nombre del archivo que contiene la función con su camino (*path*) si fuera necesario. El siguiente ejemplo muestra cómo incluir el archivo *funciones.sh* y cómo utilizar la función *espacio_ocupado* que acabamos de diseñar. El objetivo es crear un script llamado *espacio.sh* que reciba como argumento el nombre lógico de una partición y muestre un mensaje por pantalla informando del espacio ocupado en dicha partición.

```
$ cat espacio.sh
#!/bin/bash
. ./funciones.sh

partición $1
espacio_ocupado
echo La partición $1 tiene ocupados $ESPACIO Kb
$
```

Ejercicios

1. Realice un programa de *shell* que reciba desde la línea de órdenes tres palabras y se encargue de mostrarlas por pantalla ordenadas alfabéticamente.
2. Repita el ejercicio anterior, pero leyendo las tres palabras de forma interactiva.
3. Realice un programa de *shell* que reciba desde la línea de órdenes dos palabras y nos indique si son iguales o distintas. Si el número de parámetros no es correcto, se deberá visualizar un mensaje de error.
4. Realice un programa *shell* que reciba desde la línea de órdenes los nombres de dos programas ejecutables. Si tras la ejecución del primero se detecta algún error, el segundo no se deberá ejecutar. Tenga en cuenta los posibles errores e indique, si se produce alguno, de qué tipo es.
5. Realice un programa de *shell* que reciba desde la línea de órdenes los nombres de dos archivos ordinarios y nos diga cuál de ellos tiene mayor tamaño. Si el número de parámetros no es correcto, se deberá visualizar un mensaje de error, así como si ambos archivos no son ordinarios.
6. Realice un programa de *shell* que pida por teclado una cadena de caracteres y no finalice hasta que la cadena sea *fin*.
7. Realice un programa de *shell* que elimine todos los archivos del directorio especificado desde la línea de órdenes y cuyo primer carácter sea la letra *a*.
8. Realice un programa de *shell* que busque en todo el disco los archivos indicados desde la línea de órdenes.
9. Realice un programa de *shell* que muestre un menú de opciones. Con la primera, enviaremos correo a un usuario que debe ser especificado. Con la segunda, se nos permitirá editar cualquier archivo de texto. Con la tercera, podremos imprimir un archivo de texto por pantalla, con la cuarta y última, podremos abandonar el programa.

Administración de sistemas operativos

Práctica 1: Lenguaje *awk*

OBJETIVOS

- Familiarizarse con el uso de expresiones regulares y utilizarlas en conjunto con el lenguaje *awk*.
- Estudiar los elementos básicos del lenguaje *awk*.
- Desarrollar programas en lenguaje *awk*, para utilizarlos como herramientas particulares dentro del sistema.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de dos sesiones de laboratorio, cada una de dos horas para un total de cuatro horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

COMPLEMENTARIA

El lenguaje de programación AWK/GAWK

Dirección: http://www.loquefaltaba.com/documentacion/Manual_Awk_castellano.pdf

Lenguaje *awk*

Dirección: http://pisuerga.inf.ubu.es/cgosorio/ALeF/Lenguaje_awk.pdf

The *awk* Manual

Dirección: <http://people.cs.uu.nl/piet/docs/nawk/nawkUS.pdf>

PRÁCTICA 1

Lenguaje *awk*

TABLA DE CONTENIDOS:

Introducción.....	265
Expresiones regulares.....	266
Otros filtros.....	270
<i>cut</i>	270
<i>tr</i>	271
El lenguaje de procesamiento <i>awk</i>	273
<i>awk</i>	273
Patrones de <i>awk</i>	274
Operadores empleados en <i>awk</i>	276
Matrices con <i>awk</i>	277
Matrices asociativas con <i>awk</i>	277
Variables mantenidas por <i>awk</i>	277
Sentencias de control de flujo.....	278
Ejecución condicional con <i>if</i>	278
Bucles con <i>while</i>	278
Bucles con <i>do</i>	278
Bucles con <i>for</i>	278
Ruptura de bucles con <i>break</i>	279
Forzar la evaluación de la condición de un bucle con <i>continue</i>	279
Finalizar la ejecución con <i>exit</i>	279
Órdenes de entrada salida.....	279
<i>print</i>	279
<i>printf</i>	279
Funciones numéricas.....	280
Funciones de tratamiento de cadenas.....	280
Ejemplos de aplicación.....	280
Ejercicios.....	283

Introducción

awk es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean ficheros o flujos de datos. *awk* es ejemplo de un lenguaje de programación que usa ampliamente el tipo de datos de listas asociativas (listas indexadas por cadenas clave) y expresiones regulares.

awk puede usarse para buscar un nombre particular dentro de un archivo o para añadir un nuevo campo a una base de datos pequeña. También se puede utilizar para realizar el tipo de funciones que proporcionan muchas de las otras herramientas del sistema Unix. Pero puesto que también es un lenguaje de programación, resulta más potente y flexible que cualquiera de ellos.

awk está especialmente diseñado para trabajar con archivos estructurados y patrones de texto. Dispone de características internas para descomponer líneas de entradas en campos y compara estos campos con patrones que se especifiquen. Debido a estas posibilidades, resulta particularmente apropiado para trabajar con archivos que contienen información estructurada en campos, como inventarios, listas de correos y otros archivos de base de datos simples.

Debido a que las expresiones regulares representan gran parte de la base de *awk*, en esta práctica empezaremos explicando todo sobre las expresiones regulares y una vez entendidas éstas y sus distintas formas de uso entraremos de lleno con el lenguaje *awk*. El cual nos va a permitir modificar archivos, buscar y transformar bases de datos, generar informes simples y otras muchas cosas.

Expresiones regulares

Una expresión regular es un patrón que define a un conjunto de cadenas de caracteres. Las expresiones regulares se construyen de forma análoga a las expresiones aritméticas. Existe la posibilidad de combinar expresiones simples; para ello, debemos emplear distintos operadores.

Los bloques básicos de construcción son las expresiones regulares que referencian un único carácter. La mayoría de los caracteres, incluyendo todas las letras y dígitos, son expresiones regulares que se definen a sí mismos. Cualquier metacarácter con significado especial debe ser precedido del símbolo *backslash* \ para que pierda su significado especial.

Una lista de caracteres encerrados dentro de [y] referencia cualquier carácter sencillo de esa lista. Si el primer carácter de la lista es un ^ (acento circunflejo), entonces estaremos haciendo referencia a los caracteres que no aparecen en la lista. Por ejemplo, la expresión regular [0123456789] representa cualquier dígito simple. Para referenciar un rango determinado de caracteres ASCII, pondremos el primero y el último de ellos encerrados entre corchetes y separados por un guión. Por ejemplo, la expresión regular [a-z] representa cualquier letra minúscula. El punto (.) representa cualquier carácter, excepto el carácter de nueva línea.

Los caracteres ^ y el \$ son metacaracteres que representan una cadena vacía al principio y al final de la línea, respectivamente. Los símbolos \< y \> representan una cadena vacía al principio y al final de una palabra.

Una expresión regular que representa un carácter sencillo puede ser continuada con uno o varios caracteres de repetición:

- ? El elemento precedente es opcional y debe coincidir al menos una vez.
- * El elemento precedente debe coincidir cero o más veces.
- + El elemento precedente debe coincidir una o más veces.
- {n} El elemento precedente debe coincidir exactamente n veces.
- {m} El elemento precedente es opcional y debe coincidir al menos m veces.
- {n,m} El elemento precedente debe coincidir al menos n veces, pero no más de m.

Las expresiones regulares pueden ser concatenadas. El resultado de la concatenación representa aquellas cadenas que concatenadas responden al patrón propuesto de expresiones regulares.

Dos expresiones regulares pueden unirse con el operador /. La expresión regular resultante representa cualquier cadena que responda al patrón de cualquiera de las dos expresiones regulares.

La operación de repetición tiene precedencia sobre la operación de concatenación. Se pueden utilizar paréntesis si queremos modificar las precedencias.

Los metacaracteres ?, +, {, }, /, (y) tienen que ser precedidos del símbolo *backslash* \ para que pierdan su significado especial.

A continuación vamos a poner una serie de ejemplos de uso de expresiones regulares. En el lado izquierdo pondremos la expresión regular (patrón), y en el derecho, su significado.

Patrón	Qué representa
gato	La cadena gato
^gato	La cadena gato al comienzo de una línea
gato\$	La cadena gato al final de una línea

Patrón	Qué representa
<code>^gato\$</code>	La cadena gato formando una única línea
<code>gat[ao]</code>	Las cadenas gata o gato
<code>ga[^aeiou]o</code>	La tercera letra no es una vocal minúscula
<code>ga.o</code>	La tercera letra es cualquier carácter
<code>^....\$</code>	Cualquier línea que contenga cuatro caracteres cualquiera
<code>^\.</code>	Cualquier línea que comienza por punto
<code>^[^.]</code>	Cualquier línea que no comienza por punto
<code>gatos*</code>	gato, gatos, gatoss, gatoss, etc
<code>"gato"</code>	gato entre comillas dobles
<code>"*gato"*</code>	gato con o sin comillas dobles
<code>[a-z][a-z]*</code>	Una o más letras minúsculas
<code>[a-z]+</code>	Lo mismo que lo anterior (sólo valido en algunas aplicaciones)
<code>^[^0-9A-Z]</code>	Cualquier carácter que no sea ni número ni letra mayúscula
<code>[A-Za-z]</code>	Cualquier letra, sea mayúscula o minúscula
<code>[Ax5]</code>	Cualquier carácter que sea A, x o 5
<code>gato gota gata</code>	Una de las palabra gato, gota o gata
<code>(s arb)usto</code>	Las palabras susto o arbusto
<code>ga?t[oa]</code>	gato, gata, gasto, gaita, etc
<code>\<ga</code>	Cualquier palabra que comience por ga
<code>to\></code>	Cualquier palabra que termine en to
<code>\<gato\></code>	La palabra gato
<code>o{2,}</code>	Dos o más o en una misma fila

Como ejemplo de aplicación de expresiones regulares vamos a estudiar el filtro *grep*. Vamos a hondar en el uso de *grep* para buscar palabras dentro de un archivo haciendo uso de las expresiones regulares.

Siempre que empleemos expresiones regulares con *grep*, deben ser encerradas entre comillas dobles para que el intérprete de órdenes no las interprete. Si dentro de la expresión regular tenemos el metacarácter `$`, deberemos emplear comillas simples en lugar de las comillas dobles.

A continuación vamos a poner una serie de ejemplos haciendo uso de *grep* y de expresiones regulares conjuntamente. Con ello, pretenderemos dejar mas claro el uso de las expresiones regulares. Para ello, vamos a trabajar con un archivo denominado *datos*, cuyo contenido es el siguiente:

```
$ cat datos
```

```
gato      libro      atunn      gotas     a         tas
pez       gaita      ##%%      dado              oso
.exrc     expresoatun      gota      loco
GAto     tierra     Gata      nada      raton
gata     canica     atunnn    fuente     gatos
fin
$
```

En primer lugar, vamos a buscar la palabra *gato* en el archivo *datos*. Los resultados se muestran seguidamente:

```
$ grep "gato" datos
```

```
gato      libro      atunn      gotas     atas
gata     canica     atunnn    fuente     gatos
$
```

Ahora buscaremos las líneas del archivo *datos* que comienzan con la palabra *gato*:

```
$ grep "^gato" datos
gato      libro      atunn      gotas      atas
$
```

A continuación visualizaremos las líneas del archivo *datos* que contienen las palabras *gato* o *gata*.

```
$ grep "gat[oa]" datos
gato      libro      atunn      gotas      atas
gata      canica     atunnn     fuente     gatos
$
```

En el siguiente ejemplo buscaremos las líneas del archivo *datos* que contienen únicamente tres caracteres.

```
$ grep "^...$" datos
fin
$
```

Seguidamente visualizaremos las líneas que contienen secuencias de una o más letras mayúsculas.

```
$ grep "[A-Z][A-Z]*" datos
GAto      tierra     Gata      nada      raton
$
```

Para ver las líneas del archivo *datos* que comienzan por punto, emplearemos la siguiente orden:

```
$ grep "^\\." datos
.exrc     expresoatun      gota      loco
$
```

Si ahora queremos ver las líneas que no comienzan por punto, utilizaremos esta otra orden:

```
$ grep "^[^.]" datos
gato      libro      atunn      gotas  a      tas
pez       gaita     ##%%      dado   oso
GAto      tierra     Gata      nada   raton
gata      canica     atunnn     fuente  gatos
fin
$
```

En el siguiente ejemplo visualizaremos las líneas del archivo *datos* que terminan en el carácter *n*. Obsérvese que se emplean comillas simples en lugar de comillas dobles con objeto de que el carácter *\$* (que indica el final de la línea) pierda su significado especial.

```
$ grep 'n$' datos
GAto      tierra     Gata      nada      raton
fin
$
```

Para visualizar las líneas que contienen tres o más *enes* seguidas, emplearemos la orden siguiente:

```
$ grep "n\{3,\}" datos
gata          canica          atunnn          fuente          gatos
$
```

Por último, si queremos ver las líneas que contienen la secuencia de caracteres en la que tenemos en primer lugar una *a*, a continuación cualquier carácter y por último una *o*, tendremos que emplear una orden como la que figura a continuación:

```
$ grep "a.o" datos
gato          libro          atunn          gotas a          tas
pez           gaita          ##%%          dado          oso
GAto         tierra          Gata          nada          raton
gata          canica          atunnn          fuente          gatos
$
```

La orden *grep* puede ser utilizada también haciendo uso de tuberías. Por ejemplo, quisiéramos visualizar los directorios del directorio */usr*, tendríamos que emplear una orden como la siguiente:

```
$ ls -l /usr | grep "^d"
drwxr-xr-x  2    root root          69632  2008-05-22 12:27 bin
drwxr-xr-x  2    root root          4096   2007-04-15 05:52 games
drwxr-xr-x  37    root root          4096   2008-05-13 12:40 include
dr-xr-x---  8    root plugdev 4096   2008-04-06 08:42 julioc
drwxr-xr-x 181    root root          65536  2008-05-22 12:27 lib
drwxr-xr-x  10    root root          4096   2007-04-15 05:48 local
drwxr-xr-x  2    root root          12288  2008-05-22 04:42 sbin
drwxr-xr-x 317    root root          12288  2008-05-22 12:27 share
drwxrwsr-x  5    root src          4096   2008-04-24 10:06 src
drwxr-xr-x  3    root root          4096   2007-04-15 05:50 X11R6
$
```

Ha que tener en cuenta que las líneas correspondientes a un directorio visualizadas por la orden *ls -l* siempre comienzan con el carácter *d*.

En el ejemplo siguiente visualizaremos los archivos ejecutables del directorio */bin* que terminan en *s*.

```
$ ls -lF /bin | grep 's*$'
-rwxr-xr-x  2    root root          1297   2008-03-20 17:49 bzless*
-rwxr-xr-x  1    root root          35412  2007-01-22 09:53 loadkeys*
-rwxr-xr-x  1    root root          77844  2007-03-05 00:25 ls*
-rwxr-xr-x  1    root root          65608  2007-03-07 15:42 ps*
-rwxr-xr-x  1    root root          11028  2007-03-04 23:43 run-parts*
-rwxr-xr-x  1    root root          54196  2007-01-16 08:19 uncompress*
-rwxr-xr-x  1    root root          1456   2007-01-16 08:19 zless*
$
```

La opción *-F* en la orden *ls* la empleamos para determinar cuáles son los archivos ejecutables. Con esta opción a los archivos ejecutables se les añadirá un asterisco al final en la visualización.

Otros filtros

cut

Sintaxis: `cut -c lista [archivo (s)]`
`cut -f lista [-d char] [archivo (s)]`

El filtro *cut* se usa para cortar y pasar a la salida estándar las columnas o campos de la entrada estándar o del archivo especificado. La opción `-c` es para cortar columnas y `-f` para cortar campos. Al cortar un campo, existe la opción `-d` para especificar los caracteres de separación entre los distintos campos (el delimitador). Por defecto, este delimitador es el tabulador, a menos que se indique otra cosa. Para especificar las columnas o campos que deseamos cortar se utiliza una lista. Una lista es una secuencia de números que se usa para indicarle a *cut* qué campos o columnas se quieren cortar. Hay varios formatos para esta lista:

- **A-B** Campos o columnas desde A hasta B inclusive.
- **A-** Campo o columna A hasta el final de la línea.
- **A, B** Campos o columnas A y B.

Para mostrar con un ejemplo el uso de *cut*, imaginemos que tenemos un archivo llamado *personas* con el siguiente contenido:

```
$ cat personas
SSP : 908732124
ASF : 456789212
MBV : 432765433
ASH : 423562563
JPA : 798452367
$ cut -c 1-3 personas
SSP
ASF
MBV
ASH
JPA
$
```

Al cortar por caracteres desde la columna 1 a la 3, nos estamos quedando con las tres primeras letras de cada línea del archivo.

Veamos otro ejemplo que combina el uso de *grep* con *cut* para obtener el listado de los usuarios del sistema que emplean el intérprete de órdenes *bash*.

1. Obtener todos los usuarios del sistema. Emplearemos la orden:

```
$ cat /etc/passwd
```

2. La salida de la orden anterior la filtraremos para obtener todas las líneas que contengan el patrón *bash*. Mediante la orden:

```
$ cat /etc/passwd | grep "bash"
```

3. Finalmente y teniendo en cuenta que el carácter delimitador de campos en el archivo */etc/passwd* es `:`, haciendo uso de *cut* nos quedaremos únicamente con los campos 1 y 7. El resultado de la ejecución de la orden podría ser algo como lo siguiente:

```
$ cat /etc/passwd | grep "bash" | cut -f 1,7 -d ':'
root:/bin/bash
gateway:/bin/bash
```

tr

Sintaxis: `tr [-dsc] cadena1 cadena2`

La orden `tr` se emplea como traductor (*translator*). Como todo filtro, `tr` lee datos en la entrada estándar, los procesa y deposita los resultados en la salida estándar. El empleo más evidente de `tr` es como conversor de letras mayúsculas a minúsculas, y viceversa. Supongamos que tenemos un archivo denominado *fich* con el siguiente contenido:

```
$ cat fich
Este es un archivo de texto
QUE CONTIENE LETRAS MAYUSCULAS Y minusculas.
$
```

A este archivo vamos a aplicarle la orden `tr` con diversas opciones.

Ejemplos:

```
$ tr [A-Z] [a-z] < fich
este es un archivo de texto
que contiene letras mayusculas y minusculas.
$
```

En el ejemplo anterior hemos convertido todos los caracteres del rango de la *A* a la *Z* en sus correspondientes del rango de la *a* a la *z*. Vamos a realizar ahora el proceso inverso, convertir de minúsculas a mayúsculas. Para ello, emplearemos la orden siguiente:

```
$ tr [a-z] [A-Z] < fich
ESTE ES UN ARCHIVO DE TEXTO
QUE CONTIENE LETRAS MAYUSCULAS Y MINUSCULAS.
$
```

También podemos sustituir un rango de caracteres por un carácter cualquiera de la forma siguiente:

```
$ tr [A-Z] x < fich
xste es un archivo de texto
xxx xxxxxxxx xxxxxx xxxxxxxxxxxx x minusculas.
$
```

En el caso anterior, hemos convertido el rango de caracteres de la *A* a la *Z* por el carácter *x*. `tr` puede ser empleado también para eliminar determinados caracteres de un archivo. Para ello, debemos emplear la opción `-d` y a continuación indicarle el carácter o caracteres que deseamos eliminar.

```
$ tr -d [A-Z] < fich
ste es un archivo de texto
minusculas.
$
```

En el caso anterior eliminamos cualquier carácter del archivo *fich* que esté comprendido en el rango *A-Z*. Podemos hacer lo mismo, pero eliminando las minúsculas.

Otra de las opciones de *tr* es la posibilidad de eliminar caracteres repetidos en el texto. Para ello, debemos emplear la opción *-s*. Supongamos que tenemos un archivo denominado *otro* con el siguiente contenido:

```
$ cat otro
Aqquiiii tteeeennnngggoooo rreeepppppeeeettiddoooss
ccciieeerrrtooss ccaaaaaaarraaacctteerrreesss
$
```

Para eliminar caracteres repetidos, haremos lo siguiente:

```
$ tr -s [a-z] < otro
Aquí tenemos repetidos
ciertos caracteres
$
```

Por último, la opción *-c* se puede emplear para indicar el complemento de un patrón de caracteres.

Ejemplo:

```
$ tr -c [A-Z] “ ” < fich
E          QUE CONTIENE LETRAS MAYUSCULAS Y
$
```

En el ejemplo anterior hemos sustituido todo carácter que no pertenezca al patrón *[A-Z]* por un espacio en blanco.

Veamos un ejemplo completo, desarrollado paso a paso, en el que localicemos todos los archivos del directorio *HOME* de un usuario que no pertenezca a dicho usuario.

1. El listado de todos los archivos los obtendremos con la siguiente orden:

```
$ ls -lR
```

Inicialmente no podremos emplear el espacio en blanco como delimitador porque se encuentra repetido en muchos puntos.

2. Para eliminar los espacios en blanco repetidos y así poder utilizar los espacios en blanco como delimitadores de campos utilizaremos la orden:

```
$ ls -lR | tr -s ‘ ’
```

3. Seguidamente tendremos que eliminar toda la información que *ls -lR* genera y que no corresponde a información de archivos. Todas las líneas que son archivos obedecen a un patrón que comienza por un carácter, que determina el tipo de archivo, seguido de un guión o *r*, de nuevo guión o *w* y por último guión o *x*. Para eliminar todo lo que no comience con el patrón indicado, emplearemos la orden:

```
$ ls -lR /usr | tr -s ‘ ’ | grep ‘^[r-] [w-] [x-]’
```

4. Finalmente eliminamos todo lo que no contenga el nombre del usuario con la orden:

```
$ ls -lR $HOME | tr -s ‘ ’ | grep ‘^[r-] [w-] [x-]’ | grep -v $USER
```

Hemos empleado la variable *\$USER* que almacena el nombre del usuario y el modificador *-v* que invierte el sentido de la búsqueda: en vez de buscar el patrón *\$USER* buscar las líneas que no contengan ese patrón.

El lenguaje de procesamiento *awk*

awk

Sintaxis: `awk [op] [Ffs] ord [-v var=val] archivo(s)`
`awk [op] [-Ffs] -f f_ord [-v var=val] archivo(s)`

Como podemos observar tenemos dos modos diferentes de invocar el programa. En el primer modo le damos las órdenes desde la propia línea de órdenes, y en el segundo (opción *-f*), le especificamos un archivo en donde se encuentran las órdenes que *awk* tiene que ejecutar. Este segundo modo es más cómodo si el conjunto total de órdenes es amplio. *awk* puede trabajar con varios archivos a un tiempo. Si no se le especifica ningún archivo, *awk* leerá en la entrada estándar. *awk* procesa los archivos especificados línea por línea, a cada línea se le compara con un patrón, y si coincide, se llevan a cabo sobre ellas las acciones que indiquemos.

awk admite las siguientes opciones, la cuales deben estar disponibles en cualquier versión del programa.

- **-Fs** Con esta opción indicaremos que el separador de campos es el carácter *s*. Esto es lo mismo que activar la variable predefinida *FS*. Por defecto, los separadores de campos utilizados por *awk* son los espacios en blanco y los tabuladores. Cada uno de los campos de una línea del archivo que se procesa puede ser referenciado por las variables *\$1*, *\$2*, ..., *\$NF*. La variable *NF* (*Number Fields*) indica el número de campos de la línea que se está procesando. La variable *\$0* se refiere a la línea completa.
- **-v var=val** Asigna el valor *val* a la variable *var* antes de que se comience la ejecución del programa. Esta asignación de variables también se puede llevar a cabo en el bloque *BEGIN* de un programa *awk*.
- **-ff_ordenes awk** Leerá las órdenes en el archivo *f_ordenes*.

Las órdenes de *awk*, como indicamos previamente, son secuencias de patrones y acciones:

patrón {acción}

Tanto el patrón como la acción son opcionales. Si falta el patrón, la acción o procedimiento se aplicará a todas las líneas. Si falta la acción, simplemente se visualizará la línea.

Vamos a ver un primer ejemplo de uso de *awk*. Para ello, vamos a procesar lo que la orden *date* envía a la pantalla, que es algo como lo siguiente:

```
$ date
dom jun 20 20:07:00 CEST 2004
$
```

Lo único que vamos a hacer es visualizar los campos primero (día), segundo (mes) y sexto (año). La forma de hacerlo es la siguiente:

```
$ date | awk '{print $1; print $2; print $6}'
dom
jun
2004
$
```

Seguidamente vamos a visualizar las líneas del archivo `/etc/passwd` que comienzan con el carácter `d`:

```
$ awk '/^d/' /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
dhcp:x:100:101::/dev/null:/bin/false
$
```

Como no hemos especificado ninguna opción, `awk` simplemente visualiza la línea que cumple el patrón que hemos indicado. El patrón anterior es una expresión regular, pero, como veremos en el punto siguiente, `awk` permite utilizar otros tipos de patrones.

Patrones de `awk`

Los patrones que `awk` reconoce pueden ser cualquiera de los siguientes:

- **BEGIN**
- **END**
- **/expresiones regulares/**
- **expresiones relacionales**
- **expresiones coincidencia de patrones**

`BEGIN` y `END` son dos tipos de patrones especiales. El patrón `BEGIN` permite especificar una serie de procedimientos que se ejecutarán antes de que ninguna línea de ningún archivo sea procesada. Generalmente, con este patrón se declaran las variables globales. El patrón `END` permite especificar los procedimientos que no queremos que se ejecuten hasta que se terminen de procesar todas y cada una de las líneas de un archivo.

Para los patrones `/expresiones regulares/`, la acción se ejecuta para cada línea que verifica la expresión regular. Estas expresiones regulares son las mismas que hemos visto anteriormente.

Las expresiones relacionales pueden utilizar cualquiera de los operadores que se definirán más tarde en el punto dedicado a ellos. Estos operadores se emplean para comprobar si algún campo verifica alguna condición. Por ejemplo, `NF > 2` selecciona las líneas en las que el número de campos es mayor que dos.

Las expresiones de coincidencia de patrones utilizan los operadores `~` (coincide) y `!~` (no coincide) para determinar si se lleva o no a cabo la acción.

Excepto para los patrones `BEGIN` y `END`, todos los patrones pueden ser combinados con operadores de *Boole*. Estos operadores son el *AND* lógico, `&&`, el *OR* lógico, `//`, y el *NOT* lógico, `!`.

Con objeto de aclarar los conceptos mostrados, vamos a poner unos ejemplos de uso de patrones. En el primer ejemplo vamos a introducir todas las órdenes dirigidas a `awk` en un archivo, y a continuación lo procesaremos. El contenido del archivo es el siguiente:

```
$ cat ejemplo1.awk
# Inicialización se ejecuta al comenzar
BEGIN{
    FS=".";
    x=0;
}
# Si la primera línea comienza con P, se visualiza el primer campo
/^P/{
```

```

        print $1;
    }
    # Si el número de campos es mayor que tres
    # visualizamos el campo cuatro
    NF > 3{
        print $4;
    }
    # Si el cuarto campo es mayor que 10, incrementamos x
    $4 > 10{
        x++;
    }
    # Finalización se ejecuta al finalizar
    END{
        print x;
    }
}
$

```

Todas las líneas que comienzan por el carácter # serán ignoradas por *awk* en el procesamiento. Así pues, podemos emplear este carácter como inicio de una línea de comentarios. En el ejemplo anterior los comentarios son explicativos de lo que hace cada línea. El archivo anterior no tiene ninguna utilidad, se ha empleado con el objetivo de mostrar el uso de patrones.

A la hora de procesar este archivo, debemos emplear la siguiente sintaxis:

\$ awk -f ejemplo1.awk archivo(s)

Veamos otro ejemplo empleado para visualizar los directorios cuyos nombres comienzan con letra mayúscula. En el ejemplo primero tenemos que seleccionar las líneas que visualiza *ls -l* que comienzan con el carácter *d* (directorios), y cuyo campo noveno (nombre del archivo) comience con letra mayúscula. Para especificar ambas condiciones, emplearemos el operador *&&* (AND lógico).

```

$ ls -l /usr | awk '$1 ~ /^d/ && $9 ~ /[A-Z]/'
drwxr-xr-x  8   root   root   4096  abr 3 21:32  X11R6
$

```

Según se puede apreciar, estamos empleando también expresiones de coincidencia de patrones. La primera expresión indica si *\$1* coincide con el patrón especificado por la expresión regular *^d/*. La segunda expresión indica si *\$9* coincide con el patrón especificado por la expresión regular */[A-Z]/*.

La forma que tiene *awk* de ejecutar los programas es la siguiente. Primero, *awk* compila el programa y genera un formato interno. A continuación, se realizan las asignaciones especificadas por medio de la opción *-v*. Seguidamente, *awk* ejecuta el código incluido en el bloque *BEGIN*, si es que existe tal bloque. Después, se procesa línea por línea el archivo o los archivos especificados en la línea de órdenes. Si no le especificamos ninguno, *awk* leerá en al entrada estándar. Una vez procesadas todas las líneas, se ejecuta el código incluido en el bloque *END*, si es que existe.

Operadores empleados en awk

Ya hemos indicado previamente que con *awk* podemos emplear distintos operadores. Éstos son los que se indican seguidamente:

- =, +=, -=, *=, /=, %= y ^= Operadores de asignación. Se admite tanto la asignación absoluta (variable = valor) como la que utiliza un operador (el resto de los modos). Como ejemplo del primer tipo de asignación, podemos poner el siguiente:
`datos = datos + $2`

Esto podría haberse hecho de una forma más compacta usando el operador +=, tal y como se muestra a continuación:

```
datos += $2
```

Este segundo caso es idéntico al primero en cuanto a funcionalidad se refiere, pero es más compacto. Los operadores +, -, *, /, % y ^ significan *suma*, *resta*, *multiplicación*, *división*, *resto de la división entera* y *exponenciación*, respectivamente.

- ? Es igual a la expresión condicional empleada en el lenguaje C. Su formato es el siguiente:
`expr1 ? expr2 : expr3`

Esto debe entenderse como sigue: si *expr1* es cierto, el valor de la expresión es *expr2*; de otro modo, será *expr3*. Sólo se evalúa *expr2* o *expr3*.

- // OR lógico.
- && AND lógico.
- ~ !~ Coincidencia y no coincidencia de expresiones regulares.
- < > <= >= != == Operadores relacionales.
- **Espacio en blanco** Concatenación de cadenas.
- + - Suma y resta.
- * / % Multiplicación, división y módulo (resto de la división entera).
- + - ! Más unario, menos unario y negación lógica.
- ^ Exponenciación.
- ++ -- Incremento y decremento, tanto en forma de prefijo como de sufijo.
- \$ Referencia a campo.

Veamos algunos ejemplos con estos operadores:

Vamos a calcular el tamaño medio de los archivos de un directorio. Para realizar esta operación introduciremos a nuestro filtro *awk* el resultado de la orden *ls -l*. El archivo de órdenes *awk* lo denominaremos *tamano.awk* y su contenido es el siguiente:

```
BEGIN{
    TAMANO = 0;
}
/^[.-r]/{
    TAMANO = TAMANO + $5;
    print "Procesado procesado " $9 " - " NR " Tamano= " $5 " Acumulado= "
TAMANO;
}
END{
    print "Tamano medio= " TAMANO/NR;
}
```

Matrices con *awk*

awk nos permite trabajar con matrices. Si a la matriz le denominamos *datos*, la forma de referenciar cada uno de los elementos consistirá en utilizar el nombre de la matriz y a continuación, entre corchetes, el número de elemento. De este modo, *datos[34]* es el elemento número 34 de la matriz. Vamos a poner un ejemplo en el que almacenamos el campo número nueve de cada línea del archivo de entrada en una matriz *a*. Para finalizar, visualizaremos toda la matriz. El programa que debemos emplear es el siguiente:

```
$ cat matriz.awk
# Almacena el campo nueve en una matriz
# Visualiza la matriz
{
    a[NR] = $9;
}
END{
    for(i=1; i<NR; i++)
        print a[i];
}
$
```

Matrices asociativas con *awk*

Las matrices de *awk*, a diferencia de las proporcionadas por otros lenguajes de programación, son asociativas. Esto significa que el elemento que utilizamos como índice no tiene que ser numérico, sino que puede ser de cualquier otro tipo. Pongamos el siguiente ejemplo:

```
BEGIN{
    animales["perro"]=3;
    animales["gato"]=8;
    print animales["perro"];
    print animales["gato"];
}
```

El resultado de la ejecución del programa anterior sería la visualización de los números 3 y 8, actuando como índices dentro de la matriz dos cadenas de caracteres.

Variables mantenidas por *awk*

En algún ejemplo anterior ya hemos utilizado algunas de estas variables, por ejemplo *NF*, *FS*, *\$0*, etc. A continuación vamos a dar un listado más completo de estas variables.

- **FILENAME** Es el nombre del archivo que está siendo procesado. Si no se ha especificado ningún archivo desde la línea de órdenes, el valor de esta variable será – (entrada estándar).
- **FNR** Es el número de línea del archivo que está siendo procesado.
- **FS** Indica cuál es el carácter separador de campos. Por defecto, es el espacio en blanco.
- **NF** Es el número de campos presentes en la línea que está siendo procesada.
- **NR** Indica el número total de líneas que han sido procesadas.
- **OFS** Es el separador de campos para la salida. Por defecto, es el espacio en blanco.
- **ORS** Es el separador de líneas de salida. Por defecto, es el carácter de nueva línea.
- **RS** Es el separador de líneas de entrada. Por defecto, es el carácter de nueva línea.
- **\$0** Representa la línea que se está procesando.

- $\$n$ Representa el campo n de la línea que se está procesando.

Sentencias de control de flujo

awk es un autentico lenguaje de programación, y como tal es capaz de trabajar con sentencias de control de flujo. Este tipo de sentencias serán descritas a continuación.

Ejecución condicional con *if*

```
if (condición){
    orden(es);
}
[else{
    [orden(es);]
}]
```

Si la condición que se evalúa es cierta, se ejecutará la orden u órdenes colocadas después del *if*. Si la condición no es cierta, se ejecutarán las colocadas después del *else* (si es que existe). La condición puede ser cualquier expresión que utilice operadores relaciones, así como operadores de correspondencia de patrones. Si se deben ejecutar varias órdenes, tanto después del *if* como después del *else*, éstas deberán ser colocadas entre llaves.

Bucles con *while*

```
while(condición){
    orden(es);
}
```

Si se verifica la condición, se ejecutará la orden. Las posibles condiciones son las indicadas anteriormente al hablar de *if*. Si se deben ejecutar varias órdenes dentro del bucle, éstas deberán ir entre llaves.

Bucles con *do*

```
do{
    orden(es);
}while(condición);
```

En este caso se ejecuta la orden indicada dentro del cuerpo *do while*. Si al evaluar la condición ésta se verifica, se volverá a ejecutar la orden. En el caso de que queramos ejecutar varias órdenes en el cuerpo del bucle, éstas deberán ir entre llaves.

Bucles con *for*

Esta orden tiene dos modos de operar. La sintaxis del primer modo es la siguiente:

```
for(i=mínimo; i<máximo; i++){
    orden(es);
}
```

En este caso, mientras el valor de la variable i esté comprendido entre mínimo y máximo, se ejecuta la orden indicada. En el caso de especificar varias órdenes, éstas deben ir entre llaves. Para la condición de finalización del bucle ($i < máximo$), se pueden emplear otros operadores relacionales. En el campo de progreso del bucle ($i++$) se pueden emplear $++$ y $--$, tanto en forma pre como post.

El segundo modo se muestra a continuación:

```
for(elemento in matriz){
    orden(es);
}
```

En este caso, para cada elemento de la matriz se ejecuta la orden indicada. En caso de especificar varias órdenes, éstas deben ir entre llaves. Para referirnos a cada elemento de la matriz utilizaremos la expresión *matriz[elemento]*, donde elemento es el número de ítem dentro de la matriz.

Ruptura de bucles con *break*

Esta sentencia se emplea para salir de un bucle *while* o *for*. Con ella podemos evitar iteraciones en caso de detectar que un bucle no tiene sentido que continúe su repetición.

Forzar la evaluación de la condición de un bucle con *continue*

Esta sentencia nos permite pasar a procesar la siguiente iteración dentro de un bucle *while* o *for*, saltando todas las posibles órdenes posteriores dentro del bucle.

Finalizar la ejecución con *exit*

Con esta sentencia se dejan de ejecutar instrucciones y no se procesan más archivos. Sólo se ejecutarán los procedimientos indicados en el patrón *END*. Así pues, *exit* sirve para finalizar el procesamiento de archivos por parte de *awk*.

Órdenes de entrada salida

print

Sintaxis: `print [argumentos] [destino]`

Con esta orden podemos imprimir los argumentos especificados en la salida. Los *argumentos* son normalmente campos, aunque también pueden ser cualquiera de las variables de *awk*. Para visualizar cadenas literales, debemos ponerlas entre comillas dobles. Si los argumentos de *print* son separados por comas, en la salida serán separados por el carácter indicado en la variable *OFS*. Si los argumentos son separados por espacios en blanco, la salida será la concatenación de los argumentos. El parámetro destino puede ser una expresión de redirección o entubamiento. De este modo, podemos redirigir la salida por defecto.

printf

Sintaxis: `printf [formato [, expresion(es)]]`

Esta orden se utiliza para visualizar con formato las expresiones que le indiquemos. Su sintaxis es muy similar a la empleada en la función *printf* descrita en el lenguaje C. Esta orden también es capaz de interpretar secuencias de escape como el carácter de nueva línea *\n* o el tabulador *\t*. Los espacios y el texto literal que deseamos visualizar deben ir entre comillas dobles. Por cada expresión que deseamos visualizar, debe existir su correspondiente formato. Los formatos más comunes son los siguientes:

- *%s* Una cadena de caracteres.
- *%d* Un número decimal.

- **%n.mf** Un número en coma flotante con *n* dígitos enteros y *m* decimales.
- **%o** Un número en octal sin signo.
- **%x** Un número en hexadecimal sin signo.

Para aclarar un poco las cosas a continuación tenemos el siguiente ejemplo:

```
$ date | awk {printf(Año %d. \n En hexadecimal: %x \n,$6,$6)}
Año 2001.
En hexadecimal: 7d1
$
```

Funciones numéricas

- **atan2(y,x)** Devuelve el valor del *arcotangente* de *y/x* en radianes.
- **cos(x)** Devuelve el *coseno* de *x* en radianes.
- **exp(x)** Función exponencial.
- **int(x)** Trunca el número *x* a un entero.
- **log(x)** Devuelve el logaritmo neperiano de *x*.
- **rand()** Devuelve un número aleatorio comprendido entre cero y uno.
- **sin(x)** Devuelve el *seno* de *x* en radianes.
- **sqrt(x)** Devuelve la raíz cuadrada de *x*.
- **srand(x)** Permite utilizar el número *x* como nueva semilla para la generación de números aleatorios. Por defecto, se utiliza como semilla la hora actual.

Funciones de tratamiento de cadenas

- **gsub(r,s,t)** Sustituye la cadena que verifica la expresión regular *r* por la subcadena *s* en la cadena total *t*. Si *t* no se proporciona, se asume que vale *\$0*.
- **index(s,t)** Devuelve la posición de la subcadena *t* en la cadena *s*. Si la subcadena *t* no se encuentra presente en *s*, *index* devuelve cero.
- **length(s)** Devuelve la longitud de la cadena *s*. Si *s* no se especifica, se asume *\$0*.
- **match(s,r)** Devuelve la posición en *s* donde se verifica la expresión regular *r*. Si no se verifica el patrón, se devuelve cero.
- **split(s,a,r)** Divide la cadena *s* en elementos de la matriz *a* (*a[0]*, *a[1]*, ..., *a[n]*). La cadena es dividida en cada ocurrencia de la expresión regular *r*. Si *r* no está presente, se asume que el separador es *FS*. *split* devuelve el número de elementos de la matriz.
- **sprintf(fm,ex)** Formatea la lista de expresiones *ex* acorde con el formato especificado por *fm* y retorna la cadena resultante. La cadena es formateada, pero no visualizada.
- **sub(r,s,t)** Opera igual que *gsub(r,s,t)*, pero sólo se reemplaza la primera subcadena que verifica la expresión regular.
- **substr(s,i,n)** Devuelve la subcadena formada por *n* caracteres a partir de la posición *i* de la cadena original *s*. Si se omite el valor *n*, se asume que la subcadena la formarán el resto de los caracteres hasta el final de la cadena *s*.
- **tolower(str)** Devuelve la cadena resultante de convertir en minúsculas las letras formantes de la cadena *str*. Los caracteres no alfabéticos no se ven afectados.
- **toupper(str)** Devuelve la cadena resultante de convertir en mayúsculas las letras formantes de la cadena *str*. Los caracteres no alfabéticos no se ven afectados.

Ejemplos de aplicación

Seguidamente, vamos a ver una serie de ejemplos de aplicación de *awk*. Con ellos se pretende dejar claros los conceptos vistos al hablar de este lenguaje de procesamiento.

En el primer ejemplo vamos a imprimir los campos de un archivo que estén separados por el carácter `:` en orden inverso. Para ello, utilizaremos la sentencia `for`. El archivo sobre el que trabajaremos se denomina *prueba*, y su contenido es el siguiente:

```
$ cat prueba
blanco:73:Marte:1543:Manuel
verde:17:Jupiter:1968:Sebastian
azul:24:Venus:1970:Ana
rojo:35:Neptuno:1122:Javier
amarillo:135:Tierra:1234:Raul
$
```

El archivo de órdenes `awk` lo denominamos *for.awk*, y su contenido es el siguiente:

```
$ cat for.awk
BEGIN{
    FS= ":"
}
{
    for(i=NF; i>=1; i--){
        print $i, ;;
    }
    print \n;
}
$
```

Para observar lo resultado hacemos: **awk -f for.awk prueba**

En el siguiente ejemplo veremos un método sencillo que nos permite calcular el tamaño total en bytes y kilobytes de los archivos de un determinado directorio. El archivo de órdenes `awk` lo denominamos *total.awk*, y su contenido es el siguiente:

```
$ cat total.awk
# Sólo nos quedamos con los archivos ordinarios
# Cuando se visualizan con ls -l comienzan con -
/^-/{
    total=total+$5;
}
END{
    print Tamaño total en bytes:, total;
    print Tamaño total en kilobytes:, total/1024;
}
$
```

Para observar el resultado ejecutamos el siguiente comando: **ls -l | awk -f total.awk**

El próximo ejemplo puede ser utilizado para calcular la longitud media del número de caracteres de los nombres de los archivos de un directorio. El programa `awk` se denomina *longfich.awk*, y su contenido es el siguiente:

```
$ cat longfich.awk
# Calculo del número de caracteres del nombre de los
# archivo visualizados con ls -l
# Nos saltamos la primera línea
NR>1{
```

```

        print $9, tiene, length($9), caracteres;
        caracteres+=length($9);
    }
END{
    print longitud media: caracteres/(NR-1);
}
$

```

Para observar el resultado ejecutamos el siguiente comando: **ls -l | awk -f longfich.awk**

En el siguiente ejemplo vamos a calcular el mayor número de identificador de usuario que existe en el archivo */etc/passwd*. Hay que tener en cuenta que el campo de *UID* del archivo es el tercero, y que los distintos campos están separados por *:*. El programa *awk* que vamos a utilizar lo denominamos *uidmax.awk*, y su contenido es el siguiente:

```

$ cat uidmax.awk
# Calcula el UID máximo de /etc/passwd
BEGIN{
    FS=":";
    x=0;
}
$3>x{
    x=$3;
}
END{
    print x;
}
$

```

Para observar el resultado ejecutamos el siguiente comando: **awk -f uidmax.awk /etc/passwd**

Ejercicios

1. En una única línea de órdenes realice las acciones oportunas para que se visualice por pantalla el mes actual y además, que quede almacenado en un archivo denominado *mes_actual*.
2. Cree un archivo denominado *personas* que contenga los nombres, apellidos y edades de 15 personas. Liste todas las personas del archivo anterior cuya edad sea de 27 años. Liste los datos de todas aquellas personas cuyo primer apellido comience con *S*. Visualice la edad de una persona que se llame *Ana*. Ordene alfabéticamente por apellidos el archivo anterior y genere un nuevo archivo en su directorio de conexión denominado *personas.orden.alfabetico*. Ordene por edades el archivo *personas* y genere un nuevo archivo denominado *personas.orden.edad*. ¿Cuántas personas existen en el archivo *personas* cuya edad sea de 23 años.
3. ¿Cómo podríamos quedarnos solamente con la información relativa a la hora que nos visualiza *date* por pantalla?
4. Liste por pantalla únicamente los archivos ordinarios que cuelgan del directorio */usr*.
5. ¿Qué orden emplearía para visualizar en mayúsculas el contenido de cualquier archivo de texto?
6. ¿Qué orden emplearía para visualizar las líneas de cualquier archivo de texto que comience con letra mayúscula?
7. Realice un programa *awk* que visualice la cantidad de disco empleada para un determinado usuario. Si esta cantidad es mayor que 10MBytes, comuníquelo mediante un mensaje.
8. Cree un archivo compuesto por varias líneas, cada una de ellas con el siguiente formato:

Nombre Apellido1 Apellido2 Nota

La nota es un valor numérico comprendido entre cero y cien. Una vez creado este archivo, realice un programa *awk* que genere un nuevo archivo en el que el campo nota se sustituya por una de las palabras siguientes:

Suspense (si nota <60)

Aprobado (si $60 \leq \text{nota} < 70$)

Notable (si $70 \leq \text{nota} < 90$)

Sobresaliente (si nota ≥ 90)

Administración de sistemas operativos

Práctica 2: Instalación del sistema operativo Linux

OBJETIVOS

- Instalar un sistema operativo Linux.
- Realizar una correcta distribución del espacio existente en el disco duro para instalar correctamente el sistema operativo.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de dos sesiones de laboratorio, cada una de dos horas para un total de cuatro horas.

BIBLIOGRAFÍA

BÁSICA

Manual básico Ubuntu GNU/Linux

Dirección:

http://www.marblestation.com/publicaciones/Ubuntu/2.%20Breezy/breezy_es.compress.pdf

COMPLEMENTARIA

Instalación de Ubuntu

Dirección: <http://www.cuscolibreweb.org/filescl/ubuntu.pdf>

Sun xVM VirtualBox

Dirección: <http://www.virtualbox.org/download/1.6.0/UserManual.pdf>

Creando una máquina virtual en VirtualBox

Dirección:

http://www.cvirtualuees.edu.sv/file.php/66/documentos/Creando_maquinas_virtuale.pdf

PRÁCTICA 2
Instalación del sistema operativo Linux

TABLA DE CONTENIDOS:

Introducción.....	287
Ejercicio.....	288

Introducción

Esta práctica está basada en un único ejercicio que consiste en llevar a cabo todo el proceso real de instalación de una distribución Linux. Aquí pondremos en práctica todos los conceptos aprendidos en los temas teóricos estudiados hasta el momento y analizaremos cómo las distribuciones actuales nos facilitan muchas de las etapas de instalación estudiadas.

La distribución de Linux que hemos elegido para instalar en esta práctica es la distribución Ubuntu (específicamente la 7.04) por ser una de las distribuciones que actualmente es considerada como la más moderna y dinámica. El proceso de instalación de la gran mayoría de distribuciones modernas es muy similar en todas, pero debemos destacar que Ubuntu es una de las distribuciones más fáciles de instalar; ya que nos proporciona un asistente de instalación que es bastante intuitivo y sencillo de utilizar. Además tiene la gran ventaja de detectar automáticamente gran mayoría del hardware moderno y de permitirnos diversas facilidades en el proceso de instalación de software, mediante el uso del sistema de gestión de paquetes *apt*.

Ejercicio

Para el desarrollo de este ejercicio haremos uso de una máquina virtual. Utilizaremos una aplicación llamada *VirtualBox* (instalada en todas las máquinas del laboratorio), la cual nos va a permitir crear y ejecutar una máquina virtual desde nuestro sistema actual.

Sun xVM VirtualBox es un software de virtualización para arquitecturas *x86* que fue desarrollado originalmente por la empresa alemana *Innotek GmbH*, pero que paso a ser propiedad de la empresa *Sun Microsystems* en febrero de 2008 cuando ésta compró a *Innotek*. Por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como *sistemas invitados*, dentro de otro sistema operativo *anfitrión*, cada uno con su propio ambiente virtual. Por ejemplo, para nuestro caso en concreto haremos uso de *VirtualBox* para instalar la distribución Ubuntu sobre el sistema operativo instalado actualmente en los laboratorios. De esta manera estaremos evitando causar posibles daños en el sistema operativo instalado en el laboratorio (incluso la misma máquina como tal) y también tendremos un sistema instalado a nuestra medida donde nosotros mismos seremos los administradores y donde pondremos en práctica todas las recomendaciones que como buenos administradores debemos seguir (no usar a *root* para todo, usar contraseñas seguras, habilitar sólo los servicios necesarios, etc).

La práctica consiste en:

- Instalar la distribución Ubuntu sobre un disco duro que ya posee un sistema operativo de la familia de Microsoft Windows instalado. Este es un caso típico que se suele dar en la gran mayoría de procesos de instalaciones, pues son los sistemas operativos Windows los que han predominado gran parte del mercado y por ende hemos de lidiar y coexistir con ellos.
- Para la instalación será necesario hacer una distribución correcta del disco duro suministrado (por el docente) para la práctica.
- Debido a que el disco suministrado para el desarrollo de la práctica no se encuentra particionado pero sí posee gran cantidad de espacio libre para poder efectuar sobre él un determinado número de particiones, nosotros debemos ser capaces de particionar el disco y crear principalmente dos particiones. La primera será la partición del sistema de archivos *raíz /*. Y la segunda será la partición de intercambio a la cual le asignaremos 512MB de espacio. Todo ello lo realizaremos utilizando el asistente de instalación de la distribución, esto deberá ser así, ya que debemos tener en cuenta que para instalar una distribución moderna no necesitamos nada más que nuestro CD de instalación.

Al final de la instalación se deberá entregar un breve reporte donde se explique cada uno de los pasos realizados durante el proceso de instalación. Puede hacer uso de capturas de pantallas de dicho proceso y recrear mediante las imágenes todo el proceso.

Usted deberá ser capaz, de acuerdo a lo estudiado en las clases teóricas, de decidir cuál será el tamaño correcto para la partición *raíz /* que contraseña deberá utilizar, cuál será la cuenta de usuario con la que accederá al sistema, etc. Todo esto con el fin de recrear un ambiente de trabajo real en donde el administrador según sus conocimientos es capaz de tomar soluciones sobre aspectos concretos del sistema.

Administración de sistemas operativos

Práctica 3: Instalación de software

OBJETIVOS

- Demostrar que se sabe instalar un software sobre el sistema, independientemente del tipo de paquete en el que este se encuentre.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de una sesión de laboratorio, correspondiente a un periodo de dos horas.

BIBLIOGRAFÍA

BÁSICA

Escogiendo nuestros métodos (binarios vs fuentes)

Capítulo 8. Instalación de software adicional

Dirección: <http://zonasiete.org/manual/ch08s03.html>

COMPLEMENTARIA

Guía documentada para Ubuntu

Dirección: <http://www.guia-ubuntu.org/index.php?title=Portada>

PRÁCTICA 3

Instalación de software

TABLA DE CONTENIDOS:

Introducción.....	291
Administrando paquetes <i>.deb</i>	292
Usando <i>dpkg</i>	292
Instalando paquetes.....	292
Opciones de forzado.....	293
Desinstalando paquetes.....	293
Consultando la base de datos de los paquetes.....	294
Listando paquetes.....	294
Mostrando el estado de un paquete.....	295
Listando los ficheros de un paquete.....	295
Usando <i>apt-get</i>	295
Editando el fichero <i>/etc/sources.list</i>	296
Actualizando los paquetes disponibles.....	296
Instalando un paquete.....	297
Actualizando paquetes instalados.....	297
Borrando paquetes.....	297
Compilando e instalando software desde las fuentes.....	297
Usando <i>alien</i>	298
Ejercicios.....	300

Introducción

La gestión y manipulación de los paquetes es un aspecto fundamental que todo administrador de sistemas debe manejar. Un paquete es uno o varios programas, librerías o componentes de software empaquetados en un solo archivo preparado para que sea instalado e integrado en el sistema operativo. La gran mayoría de distribuciones actuales proporcionan las herramientas necesarias para poder instalar y gestionar adecuadamente estos paquetes. También, proporcionan herramientas, especialmente para los desarrolladores de software, para poder crear otros nuevos. En estos paquetes se suelen incluir los ejecutables del programa y sus dependencias y conflictos con otras aplicaciones. Las dependencias indican, al instalar un paquete, si necesitan otros programas para que la aplicación funcione correctamente, mientras que los conflictos nos informan de incompatibilidades entre programas instalados y el que queremos instalar. Los sistemas de paquetes están diseñados de esta forma para facilitar la instalación de las nuevas aplicaciones, ya que algunas librerías son utilizadas por más de un programa y no tendría sentido que todas las aplicaciones que las utilizaran las instalaran de nuevo.

En esta práctica veremos cómo está organizado el sistema de paquetes de la distribución Ubuntu por la gran cantidad de herramientas que se proporcionan y la flexibilidad de su configuración. Además, aprenderemos cómo instalar un programa a partir de su código fuente, ya que en algunos casos podemos encontrarnos que el programa que necesitamos no esté empaquetado. Esta forma de instalación era la que se utilizaba siempre antes de que aparecieran los primeros sistemas de gestión de paquetes, que surgieron para facilitar todo este proceso. También, aprenderemos a utilizar una herramienta muy importante a la hora de trabajar con paquetes, dicha herramienta es *alien*, esta nos permite hacer diversas transformaciones sobre los distintos tipos de paquetes que existen en el mundo de Linux.

Administrando paquetes *.deb*

Los paquetes *.deb*, por norma general contienen ficheros binarios para instalar así como otra información, conocida como *metadata*; este incluye información del paquete, scripts que serán ejecutados, la lista de dependencias y conflictos o sugerencias. Algunos paquetes traen el código fuente y pueden ser compilados a mano.

Para los nombres de los paquetes se suele utilizar la siguiente convención:

paquete_version-build_arquitectura.deb

- **paquete:** Es el nombre del programa o utilidad.
- **versión:** Es el número de versión de la aplicación.
- **build:** Es el número que indica la versión del paquete, cada vez que se hace un empaquetado se incrementa.
- **arquitectura:** Es la plataforma para la cual fue destinada la compilación del paquete.

Usando *dpkg*

dpkg es el núcleo del sistema de empaquetado de las distribuciones basadas en paquetes *.deb*, o distribuciones basadas en Debian, la gran mayoría de herramientas usan *dpkg* y lo hacen más sencillo o con más opciones. A veces es más rápido usar el *dpkg* que otras herramientas a priori más sencillas.

Instalando paquetes

Sintaxis: `dpkg --install <nombre_paquete>.deb`
`dpkg -i <nombre_paquete>.deb`

Durante la instalación del paquete, *dpkg* revisará si existen las dependencias necesarias para la instalación e informará con un error si no están instaladas.

Ejemplo:

```
$ dpkg -i ethereal_0.8.13-2_i386.deb
Selecting previously deselected package ethereal.
(Reading database ... 54478 files and directories currently installed.)
Unpacking ethereal (from ethereal_0.8.13-2_i386.deb) ...
dpkg: dependency problems prevent configuration of ethereal:
ethereal depends on libpcap0 (>= 0.4-1); however:
Package libpcap0 is not installed.
dpkg: error procesing ethereal (--install):
dependency problems - leaving unconfigured
Errors were encountered while processing: Ethereal
```

Como se puede observar es necesario el paquete *libpcap0*, debemos por tanto instalarlo por separado o bien con el mismo comando como sigue:

```
$ dpkg -i ethereal_0.8.13-2_i386.deb libpcap0_0.4a6-3_i386.deb
(Reading database ... 54499 files and directories currently installed.)
Preparing to replace ethereal 0.8.13-2 (using ethereal_0.8.13-2_i386.deb)
Unpacking replacement ethereal ...
Selecting previously deselected package libpcap0.
Unpacking libpcap0 (from libpcap0_0.4a6-3_i386.deb) ...
Setting up libpcap0 (0.4a6-3) ...
```

Setting up etherreal (8.13-2) ...

Opciones de forzado

En ocasiones es necesario, bien por gusto o por necesidad, sobrescribir un error cuando se instala o se borra un programa. *dpkg* ofrece varias opciones para ignorar los errores, estas se listan en la siguiente tabla:

Opción	Uso
configure-any	Configura otros paquetes que ayudarán al actual en su instalación.
hold	Procesa otro paquete, incluso si está marcado como <i>hold</i> (fijado).
bad-path	Incluso con ficheros perdidos.
not-root	Intenta eliminar o añadir paquetes aún cuando no se es <i>root</i> .
overwrite	Sobrescribe un fichero de un nuevo paquete, incluso si corresponde a otro paquete.
depends-version	Convierte un error por falta de una versión concreta en las dependencias en un <i>warning</i> , de ese modo puede continuar la instalación.
depends	Convierte todos los errores de dependencias en <i>warnings</i> .
confnew	Usa siempre el archivo de configuración más nuevo.
confold	Usa siempre el archivo de configuración más viejo.
conflicts	Permite que paquetes con conflictos sean instalados.
overwrite-dir	Sobrescribe el directorio de otro paquete por el nuevo.
remove-essential	Borra paquetes del sistema, peligroso.

Por ejemplo, si se quiere instalar un programa que tiene conflictos con otro, se debe de teclear:

```
$ dpkg -i <nombre_paquete>.deb -force-conflicts
```

Desinstalando paquetes

```
Sintaxis: dpkg --remove <nombre_paquete>
          dpkg -r <nombre_paquete>
```

Estos comandos borran todos los ficheros del paquete excepto los ficheros de configuración, que pueden ser necesarios en una posterior re-instalación. Para quitar todos los ficheros se debe usar la siguiente opción:

```
Sintaxis: dpkg --remove --purge <nombre_paquete>
          dpkg -r -P <nombre_paquete>
```

Al igual que durante la instalación de un programa, al desinstalarlo, *dpkg* comprueba las dependencias.

Ejemplo:

```
$ dpkg -r libpcap0
dpkg: dependency problems prevent removal of libpcap0:
etherreal depends on libpcap0 (>=0.4-1).
dpkg: error processing libpcap0 (--remove):
dependency problems – not removing
Errors were encountered while processing:
libpcap0
```

Consultando la base de datos de los paquetes

El sistema gestor de paquetes mantiene una base de datos donde se recopilan todos los paquetes instalados en el sistema, la herramienta *dpkg* permite consultar esa base de datos:

```
Sintaxis: dpkg --print-avail <nombre_paquete>
          dpkg -p <nombre_paquete>
```

Por ejemplo para visualizar información del paquete *ethereal* teclearíamos:

```
$ dpkg -p ethereal
```

Esto nos devolverá información sobre: Quién mantiene el paquete, su tamaño, versión, dependencias, descripción, la suma md5, etc.

Listando paquetes

```
Sintaxis: dpkg --list <patrón>
          dpkg -l <patrón>
```

Esto nos permite obtener una lista de todos los paquetes instalados en el sistema, la opción *<patrón>* es un parámetro opcional de búsqueda, sin él, se listarán todos los paquetes instalados en el sistema.

Por ejemplo, para listar todos los paquetes que tengan que ver con *apache* se introduce el comando:

```
$ dpkg -l apache*
Desired=Unknown/Install/Remove/Purge/Hold
 | Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
 |/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase.=bad)
||/ Nombre      Versión  Descripción
+++-----
=====
pn apache <none> (no description available)
pn apache-common <none> (no description available)
pn apache-dev <none> (no description available)
pn apache-doc <none> (no description available)
pn apache-modules <none> (no description available)
```

Hay varios paquetes de *apache* listados, pero ninguno de ellos instalado, sin embargo si hubo alguna vez que estuvieron instalados. Hay tres columnas a la izquierda, con el siguiente significado:

- **p**: Significa que el paquete fue desinstalado.
- **n**: Significa que no está instalado.
- **u**: Desempaquetado y listo para instalar.
- **i**: Instalado.
- **h**: Medio instalado.

Existen diversos estados en los que se pueden encontrar los paquetes, Estos estado son:

Estado de selección: se usa con el comando *dselect*, los posibles estados son:

- **unknown**: Estado desconocido.
- **install**: El paquete está marcado para su instalación.

- **remove:** Marcado para desinstalar.
- **purge:** Marcado para desinstalación completa.
- **hold:** Marcado como fijo, no será actualizado.

Estado actual:

- **not installed:** No instalado.
- **installed:** Instalado.
- **config-files:** No está instalado pero existen ficheros de configuración.
- **unpacked:** Desempaquetado y listo para instalar.
- **failed-config:** Ocurrió un problema al ejecutarse la configuración en la instalación.
- **half-installed:** La instalación no se completó.

Errores:

- **None:** No hay errores.
- **Hold:** Está marcado como estático, no puede ser borrado ni actualizado.
- **Reinstallation required:** Se requiere reinstalación del paquete.

Mostrando el estado de un paquete

Sintaxis: `dpkg --status <nombre_paquete>`

`dpkg -s <nombre_paquete>`

Esto nos permite mostrar el estado individual de cada paquete con todos los detalles del mismo.

Ejemplo:

```
$ dpkg -s etherreal
```

```
Package: etherreal
```

```
Status: install ok installed
```

```
Priority: optional
```

```
Section: net
```

```
Installed-Size: 2996
```

```
Maintainer: Frederic Peters <fpeters@debian.org>
```

```
Version: 0.8.13-2
```

```
Depends: libc6 (>= 2.1.94), libglib1.2 (>= 1.2.0), libgtk1.2 (>= 1.2.8-1), libpcap0 (>= 0.4-1), libsnmp4.1, xlibs (>= 4.0.1-1), zlib1g (>= 1:1.1.3)
```

```
Description: Network traffic analyzer Etherreal is a network traffic analyzer, or "sniffer", for Unix and Unix-like operating systems. It uses GTK+, a graphical user interface library, and libpcap, a packet capture and filtering library.
```

Listando los ficheros de un paquete

Sintaxis: `dpkg --listfiles <nombre_paquete>`

`dpkg -L <nombre_paquete>`

Esto nos permite listar los ficheros que contiene un paquete.

Usando *apt-get*

apt-get es la herramienta por excelencia para la administración de paquetes *.deb*, sin la necesidad de una interface como la de *dselect* y teniendo un abanico más amplio de opciones. *apt-get* instalará automáticamente los paquetes así como sus dependencias.

Editando el fichero */etc/sources.list*

Antes de que *apt-get* pueda coger los paquetes para su instalación, tiene que saber de donde obtenerlos. El fichero */etc/sources.list* tiene las direcciones de las ubicaciones desde donde obtener todos los paquetes. Este fichero está formado por un listado de fuentes con el siguiente formato para los binarios:

deb uri distribución componente

Y el siguiente formato para las fuentes:

deb-src uri distribución componente

URI (Uniform Resource Identifier) es un superconjunto del familiar formato *URL* que la gran mayoría conoce, usa el siguiente formato:

protocolo://host/path

La sección *//host* del *URI* solamente se usa para los métodos *HTTP* y *FTP*, los cuatro tipos de acceso son:

- **CD-ROM:** Un CD-ROM local.
- **File:** Un directorio local.
- **FTP:** Un servidor FTP.
- **HTTP:** Un servidor WEB.

La distribución que estamos utilizando en los laboratorios (Ubuntu) divide todo el software en cinco secciones, llamadas componentes; con la finalidad de reflejar las diferencias que existen en licencias y la prioridad con la que se atienden los problemas que informen los usuarios. Estos componentes son:

- **main:** Contiene solamente los paquetes que cumplen los requisitos de la licencia de Ubuntu, y para los que hay soporte disponible por parte de su equipo. Los paquetes de este componente poseen ayuda técnica garantizada y mejoras de seguridad oportunas.
- **restricted:** Contiene el programa soportado por los desarrolladores de Ubuntu debido a su importancia, pero que no está disponible bajo ninguna licencia libre para incluir en *main*.
- **universe:** Contiene una amplia gama del programa, que puede o no tener una licencia restringida, pero que no recibe apoyo por parte del equipo de Ubuntu.
- **commercial:** Contiene programas comerciales.
- **multiverse:** Contiene los paquetes sin soporte debido a que no cumplen los requisitos de software libre.

Es recomendable poner los recursos más rápidos en la parte de arriba del fichero */etc/sources.list*. Se pueden añadir comentarios a la lista usando el símbolo *#*, puede ser útil comentar los recursos por temática, paquetes concretos, etc.

Actualizando los paquetes disponibles

La base de datos contiene una lista de todos los paquetes disponibles. Es útil, y casi necesario, actualizar esta lista a menudo, o cuando se realicen cambios al fichero */etc/sources.list*. Para actualizar la base de datos se debe ejecutar el comando:

\$ sudo apt-get update

Lo cual hará que *apt-get* recorra los recursos del fichero */etc/sources.list* y actualice la base de datos.

Instalando un paquete

Cuando se ordena la instalación de un paquete, *apt-get* revisa primero si este ya fue descargado, si no lo fue, entonces *apt-get* irá al primer recurso ubicado en el fichero */etc/sources.list* a buscar la versión más nueva del programa, y si este tiene dependencias se añadirán a la lista de instalación. Para instalar un programa se debe ejecutar el comando:

```
$ sudo apt-get install <nombre_paquete>
```

Actualizando paquetes instalados

Una de las mejores características del sistema *apt-get*, es la posibilidad de actualizar todos los paquetes instalados en el sistema a la última versión disponible y en un solo paso. Para realizar esto, podemos utilizar el comando:

```
$ sudo apt-get upgrade
```

Es necesario y muy conveniente, asegurarnos de ejecutar: **sudo apt-get update** antes de teclear el comando anterior, para que de esta manera nos aseguremos que tenemos la base de datos actualizada. Dependiendo de la cantidad de programas instalados y de las novedades, el proceso llevará más o menos tiempo.

Borrando paquetes

Los paquetes pueden ser borrados con *apt-get* al igual que con *dpkg*, el comando para realizar esta operación con *apt-get* es:

```
$ sudo apt-get remove <nombre_paquete>
```

Al igual que con *dpkg*, este comando borrará todos los ficheros del paquete excepto los ficheros de configuración, que pueden ser necesarios en una posterior re-instalación. Para quitar todos los ficheros se debe usar la siguiente opción:

```
$ sudo apt-get remove --purge <nombre_paquete>
```

Compilando e instalando software desde las fuentes

En ocasiones es necesario instalar un software que no se encuentra dentro de los típicos paquetes *.deb* o *.rpm*, sino que se encuentran en formatos como *.tar.gz* o *.tar.bz2* o *.tgz*, ante estos casos estamos ante la presencia de un software que deberá ser instalado desde sus fuentes, es decir que debemos compilar e instalar por nuestra propia cuenta, sin la ayuda del sistema de gestión de paquetes propio de nuestra distribución.

Generalmente la gran mayoría de software que se proporciona de esta manera se suele instalar mediante una serie de pasos predefinidos, pero nos podemos encontrar con ciertos casos en que dichos pasos predefinidos no son válidos para la instalación, es por esto que la gran mayoría de desarrolladores de software incluyen dos archivos que siempre debemos revisar antes de instalar el software, estos archivos son *README* e *INSTALL*. En ellos se describe cuál es el procedimiento necesario para realizar una instalación correcta del software, así como también, cuáles son las dependencias necesarias para la correcta instalación del mismo.

Los pasos predefinidos que generalmente se suelen seguir para instalar un software desde sus fuentes son:

1. **\$ tar xvfz <nombre_paquete>.tar.gz**
(o **\$ tar xvfz <nombre_paquete>.tgz**)
(o **\$ tar xvfj <nombre_paquete>.tar.bz2**)
2. **\$ cd <nombre_paquete>**
3. **\$./configure**
4. **\$ make**
5. **\$ sudo make install**

El primer paso es el paso de desempaquetado. En este paso estamos descomprimiendo el contenido del paquete *<nombre_paquete>.tar.gz*, ya que la gran mayoría de paquetes de este tipo suelen venir así pues son más fáciles de transportar y además guardan una estructura de sistema de archivos interna.

Una vez que hemos descomprimido el paquete se ha creado un directorio llamado *<nombre_paquete>*, en el se encuentran todos los archivos necesarios para la compilación del software. Es por esto que es necesario que ingresemos en el nuevo directorio mediante el comando *cd*.

Ahora es el momento de configurar el paquete. Por lo general, pero no siempre (siempre leer antes los archivos *README* e *INSTALL*) se hace ejecutando el *script configure*. Cuando ejecutamos el *script configure*, aún no estamos realizando ningún proceso de compilación, sino que estamos asignando los valores para las variables del sistema de dependencias que debe cumplir el paquete, es en esta etapa donde se nos informa cuales paquetes debemos instalar para poder compilar el software. Si todo ha salido bien y cumplimos todas las dependencias estos valores de las variables se utilizan para generar un *Makefile* y el *Makefile* se utiliza a su vez para generar el binario.

Es la hora de construir el binario, el programa ejecutable, a partir del código fuente. Esto se logra ejecutando el comando *make*. Debemos tener en cuenta que es necesario haber creado el *Makefile* con éxito para poder construir el binario, de lo contrario *make* no sabrá como proceder.

Ahora estamos listos para instalar el programa. Para llevar a cabo este paso es necesario que lo hagamos como *root* y que ejecutemos *make install*

Usando *alien*

Algunos paquetes están en otros formatos diferentes a *.deb*, para solucionar este problema se creo la herramienta *alien*, que es capaz de convertir cualquiera de los siguientes formatos:

- *.deb*
- *.rpm*
- *.tgz*
- *.slp*

La sintaxis del comando es:

alien [opciones] paquete

Las opciones del comando son:

Opción	Alternativa	Uso
-d	--to-deb	Opción por defecto, se utiliza para convertir en un paquete a <i>.deb</i> .
--patch=<filename>		Solamente usado con la opción <i>-d</i> . Especifica el fichero con el <i>patch</i> que debe ser usado.
--nopatch		Sólo usado con la opción <i>-d</i> . Ningún <i>patch</i> se empleará.
-r	--to-rpm	Se utiliza para convertir un paquete a <i>.rpm</i> .
-t	--to-tgz	Se utiliza para convertir un paquete a <i>.tgz</i> .
--to-slp		Se utiliza para convertir un paquete a <i>stampede</i> .
-i	--install	Instala el programa tras la creación del paquete.
-g	--generate	Desempaqueta el contenido del paquete pero no genera ninguno nuevo.
-s	--single	Lo mismo que la opción <i>-g</i> , pero no crea el directorio <i>.orig</i> .
-c	--scripts	Incluye los <i>scripts</i> en el paquete.
-k	--keep-version	No cambia la versión del nuevo paquete.
--description=		Pone <i>descripción</i> al paquete creado.
-h	--help	Muestra la ayuda.
-v	--version	Muestra la versión.

Ejercicios

1. Active los siguientes repositorios en el archivo */etc/sources.list*:
 - Main
 - Universe
 - Restricted
 - Multiverse
2. Introduzca el disco de instalación de la distribución con la que se está trabajando en el laboratorio en la unidad lectora de CD-ROM y ejecute el comando que permita añadirlo dentro de los elementos de búsqueda del archivo */etc/sources.list*.
3. Actualice la base de datos de repositorios utilizando la herramienta *apt-get*.
4. Instale los siguientes paquetes en el sistema: *acroread* y *mozilla-acroread*. Haga uso de *apt-get*.
5. Haciendo uso de la herramienta *dpkg*, instale el paquete *volleyball_0.82-1_i386.deb*, en caso de presentar dependencias incumplidas, investigue la forma de cumplir dichas dependencias haciendo uso no de *dpkg*, sino de *apt-get*. Una vez instalado compruebe su correcto funcionamiento.
6. Investigue la forma de instalar el paquete *virtualbox* mediante el comando *apt-get*. Cabe destacar que son necesarios algunos pasos previos a la instalación de dicho paquete. Así como también la añadidura de: primero una línea dentro del archivo */etc/sources/list* y segundo una clave de acceso al repositorio.
7. Haciendo uso del comando *apt-get*, actualice toda la lista de paquetes instalados en el sistema.
8. ¿Cuál es la ventaja que ofrece instalar los paquetes desde los repositorios en vez de instalarlos utilizando la herramienta *dpkg* mediante paquetes *.deb*?
9. Instale el paquete: *emacs-21.4a.tar.gz*, siguiendo los pasos del apartado: Compilando e instalando software desde las fuentes. En caso de dependencias incumplidas resuélvalas y después de haberlo instalado correctamente compruebe la instalación. En caso de error corrija la instalación.
10. Utilizando el comando *alien*, convierta el paquete *skype-2.0.0.68-fc5.i586.rpm* a un paquete en formato *.deb*. ¿Cuál sería el comando completo para añadir los scripts en el paquete? ¿Y cuál sería el comando completo para no cambiar la versión del nuevo paquete generado con *alien*? En caso de no tener instalado el paquete *alien*, deberá ser capaz de instalarlo de acuerdo a lo estudiado en las secciones anteriores. En caso de error corrija la instalación.
11. Investigue cuál es el procedimiento que se debe seguir para instalar un paquete con la extensión *.bin*. Una vez que lo sepa, instale el siguiente paquete: *RealPlayer11GOLD.bin*. Una vez instalado compruebe su correcto funcionamiento. En caso de error corrija la instalación.

NOTA: Los paquetes necesarios para el desarrollo de algunos incisos de la práctica deberán ser solicitados a su respectivo docente.

Administración de sistemas operativos

Práctica 4: Arranque y apagado del sistema

OBJETIVOS

- Conocer la metodología básica, acerca de la inicialización y detención del sistema Linux, así como los niveles de ejecución.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de una sesión de laboratorio, correspondiente a un periodo de dos horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

COMPLEMENTARIA

Guía documentada para Ubuntu

Dirección: <http://www.guia-ubuntu.org/index.php?title=Portada>

Inicio y cierre del sistema

Dirección: http://doc.ubuntu-es.org/Inicio_y_cierre_del_sistema

PRÁCTICA 4

Arranque y apagado del sistema

TABLA DE CONTENIDOS:

Introducción.....	303
La secuencia de arranque de la ROM.....	304
La secuencia de arranque del sistema operativo.....	304
Niveles de ejecución.....	304
El archivo <i>/etc/inittab</i>	305
Gestor de arranque <i>GRUB</i>	306
Parada del sistema.....	306
shutdown.....	306
Ejercicios.....	308

Introducción

Desde que encendemos el ordenador hasta que se carga en totalidad el sistema, se ejecutan varias tareas automáticamente que se conocen con el nombre de secuencia de arranque del sistema. El proceso de arranque incluye varias comprobaciones de sanidad, y con frecuencia tratará de reparar cualquier daño encontrado, especialmente daños en el disco duro. Normalmente el proceso de arranque es más rápido si la desconexión anterior fue correcta; es decir, no se realizó de manera abrupta tras un corte del fluido eléctrico o tras el apagado directo desde el botón de encendido.

Hay dos fases en la puesta en marcha del sistema: la primera de ellas es particular para cada máquina, y la segunda es característica del sistema operativo. Ambas secuencias se les conoce como:

- Secuencia de arranque (*boot*) de la ROM.
- Secuencia de arranque del sistema operativo.

En esta práctica estudiaremos un poco cada una de las dos secuencias de arranque y analizaremos los elementos que participan en dichas secuencias. Así como también aprenderemos a configurar algunos de ellos.

La secuencia de arranque de la ROM

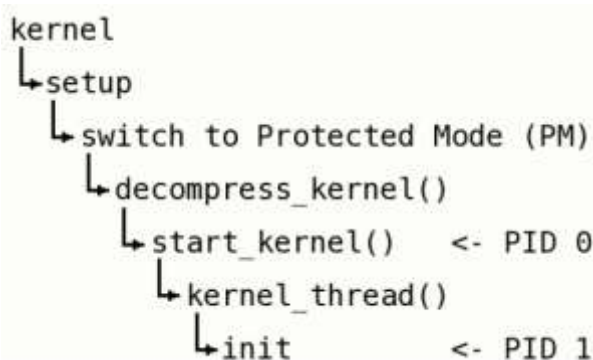
El programa de inicio de cualquier ordenador siempre está almacenado en una memoria ROM. Es en esta memoria donde el procesador comienza a leer código con objeto de ejecutarlo. Este código es característico de cada tipo de ordenador. El programa de arranque suele realizar una comprobación de todo el hardware del sistema. Si todo es correcto, lo que hará a continuación será leer del disco un programa cargador, que cargará en memoria el núcleo del sistema y finalmente le pasará el control. El archivo que contiene el núcleo del sistema normalmente se almacena en el directorio raíz del sistema de archivos y puede tener distintos nombres. Los nombres más utilizados pueden ser: *vmlinuz*, *vmUNIX*, *image* o *zimage*.

La secuencia de arranque del sistema operativo

Cuando el cargador software comienza su ejecución, muestra un mensaje similar al siguiente:

Booting Linux system

Y carga entonces el núcleo del sistema operativo en la memoria de la máquina. El cargador software cederá luego el control al núcleo recién cargado y el sistema comienza a iniciarse.



Niveles de ejecución

Los niveles de ejecución son un estado, o modo, en el que entra el sistema en el proceso de arranque y que define los servicios que serán arrancados por la máquina. Linux está programado para ejecutarse en un determinado nivel de ejecución. El número de niveles y sus nombres están predeterminados. En cambio, las acciones a realizar en cada nivel son configurables por el superusuario.

Existen siete niveles de ejecución en total:

- **Nivel de ejecución 0:** Apagado.
- **Nivel de ejecución 1:** Monousuario (sólo usuario root, no es necesaria la contraseña). Se suele utilizar para analizar y reparar problemas.
- **Nivel de ejecución 2:** Multiusuario sin soporte de red.
- **Nivel de ejecución 3:** Multiusuario con soporte de red.
- **Nivel de ejecución 4:** Como el nivel de ejecución 3 pero no se suele utilizar.
- **Nivel de ejecución 5:** Multiusuario con modo gráfico (*X Windows*).
- **Nivel de ejecución 6:** Reinicio.

Un sistema Linux no se arranca o detiene, sino que simplemente se cambia su nivel de ejecución. Durante un arranque normal, el sistema se coloca en el nivel 3 (multiusuario con red) o en el nivel 5 (análogo al 3 pero con el sistema de ventanas activo desde el inicio).

Ejemplo:

```
Cambia el nivel actual al nivel 0 (halt).
$ sudo shutdown -h now
Cambia el nivel actual al nivel 6 (reboot).
$ sudo shutdown -r now
Cambia al <nivel> especificado.
$ sudo init <nivel>
Indica el nivel de ejecución previo y el actual
$ runlevel
```

El archivo */etc/inittab*

Es el primer archivo que es leído al arrancar el sistema, contiene especificaciones sobre que otros archivos deben de ser ejecutados y el nivel de arranque del sistema, es administrado por *init*.

El formato del archivo */etc/inittab* es el siguiente:

id:nivel:acción:procesos

- **id:** Consta de uno o dos caracteres que se utilizan para identificar esa línea en el archivo.
- **nivel:** Define el nivel o niveles de ejecución para los cuales la entrada es válida. Los valores admitidos son:
 - Un número del 0 al 6 o una combinación de ellos. Se permiten valores múltiples en este campo, en cuyo caso indican que la entrada es válida para todos los niveles de ejecución listados.
 - Un campo vacío, lo cual implica que la entrada es valida para todos los niveles de ejecución de *init*.
- **acción:** Contiene una palabra clave que le dice a *init* cómo ejecutar el proceso especificado en cuarto campo. Los valores que se permiten para este campo son: *respawn, wait, once, boot, bootwait, powerfail, powerwait, off, initdefault* y *sysinit*.
- **procesos:** Contiene el proceso que se ejecutará cuando se introduzca el correspondiente nivel de ejecución.

El proceso *init* controla en todo momento el modo de funcionamiento del sistema global a partir del archivo de configuración */etc/inittab*. A continuación se muestra un ejemplo del contenido del archivo:

De forma general, existe un directorio */etc/rc<x>.d/*, por cada nivel de ejecución definido por el sistema, aquí se encuentran los servicios que deberán ser lanzados y parados en ese nivel de ejecución.

Hay que tener en consideración que los scripts que residen en el directorio */etc/init.d* pueden utilizarse directamente, lo que permite iniciar o detener servicios de forma manual. Por ejemplo, los siguientes mandatos detienen el subsistema de red y lo vuelven a iniciar:

```
$ sudo /etc/init.d/networking stop
$ sudo /etc/init.d/networking start
```

Gestor de arranque *GRUB*

GRUB es un gestor de arranque que nos permite seleccionar qué sistema operativo instalado en nuestro disco duro deseamos arrancar en el momento de arranque del sistema. Permite también que el usuario pase argumentos al *kernel*.

Dentro de sus principales características tenemos:

- Proporciona un entorno verdadero basado en comandos, lo cual supone disponer de un pre-sistema operativo en el momento del arranque.
- Soporta el modo Direccionamiento Lógico de Bloques (*LBA*). El modo *LBA* permite la conversión de direccionamiento utilizada para buscar archivos en la unidad de disco duro del firmware y se utiliza en muchos discos IDE y en todos los discos duros SCSI.
- Puede leer casi todo tipo de particiones. Esto permite que *GRUB* acceda a su archivo de configuración, */boot/grub/menu.lst*.

Presenta dos interfaces de usuario:

- Interfaz de menú
 - Permite escoger entradas que han sido definidas en el archivo de configuración de *GRUB*,
 - Permite acceder a una línea de comando para ejecutar acciones de arranque que deseemos.

Esta es la interfaz por defecto cuando se configura *GRUB* desde el programa de instalación. En esta interfaz hay un menú de sistemas operativos o *kernels* preconfigurados en forma de lista ordenada por nombre. Se puede utilizar las teclas de flecha para seleccionar una opción en lugar de la selección por defecto y pulsar la tecla *Enter* para arrancar el sistema.

- Interfaz línea de comandos
 - Al cargar busca archivo de configuración */boot/grub/menu.lst*
 - Si lo encuentra, la interfaz de menú se activa, utilizando las entradas encontradas en el archivo.
 - Si se elige la opción de menú línea de comandos o no se encuentra el archivo de configuración, entonces *GRUB* entra la interfaz de línea de comandos

La interfaz de línea de comandos nos proporciona un *prompt* parecido a una *shell*. Cada comando introducido aquí es ejecutado inmediatamente después de presionar la tecla *Enter*.

Parada del sistema

Al igual que el arranque en el momento de querer detener el sistema se puede realizar de varias formas; la primera es apagar el sistema (aunque **no es recomendable** ya que puede causar daños a los archivos), la segunda y más recomendada es utilizar el siguiente comando:

shutdown

Sintaxis: `shutdown [-rhf] [-t espera] [Mens]`

shutdown provoca el cese de toda actividad del sistema. Para poder ejecutar esta orden debemos hacerlo como administrador del sistema (*root*).

Opciones:

- **-r: *reboot***. Realiza una carga del sistema automáticamente después de la parada. Esta opción la utilizaremos cuando simplemente queramos reiniciar el sistema.
- **-h: *halt***. Desconecta el sistema después de la parada.
- **-t *seg***: Numero de segundos que debe esperar antes de realizar cualquier actividad.

Ejercicios

1. ¿Cuántas terminales virtuales tiene configuradas el sistema? ¿Cómo se accede a ellas? ¿Cómo se accede a la terminal de *X Window*?
2. Reduzca a dos el número de terminales virtuales configuradas en el sistema. ¿Es necesario reiniciar el sistema para que tengan efecto estos cambios? ¿Cómo se accede ahora a la terminal *X Window*?
3. Desactive el funcionamiento de las combinaciones de teclas *Ctrl+Alt+Supr*. Luego vuélvalo a activar.
4. ¿Cuáles son las diferentes maneras de iniciar el sistema en modo *monousuario*?, mencione todas las posibles soluciones.
5. ¿Qué orden utilizaría para cambiar al modo de ejecución tres?, mencione todas las posibles soluciones.
6. Cada fichero *script* del directorio */etc/init.d* suele admitir los parámetros *start*, *stop*, *restart* y *status* y puede ser ejecutado de forma independiente. Según esto, haga lo siguiente:
 - Compruebe el estado del demonio *atd*.
 - Si está en ejecución, párelo, de lo contrario arránquelo.
 - Vuelve a comprobar su estado.
 - A continuación lance el demonio.
7. Modifique el tiempo de espera antes del arranque del sistema por defecto, en el menú mostrado por *GRUB*. ¿Qué pasa si este valor es cero?
8. Cambie el sistema que se carga por defecto en el menú mostrado por *GRUB*.
9. Muestre la imagen: *grubcircle.xpm.gz* en el menú mostrado por *GRUB* al arrancar el ordenador.
10. Realice lo siguiente:
 - Teclea la orden *uname -r* y anote los resultados.
 - Reinicie el ordenador desde la consola.
 - Cuando aparezca el menú mostrado por *GRUB* presiona la tecla *e* para editar.
 - Diríjase a la línea del *kernel* que está utilizando el sistema (valor devuelto por el comando *uname -r*) y presiona la tecla *e*.
 - Diríjase al final de la línea y teclee lo siguiente:
rw init=/bin/bash
 - Presione la tecla *Enter* y luego la tecla *b* para reiniciar.

Indique todo lo que ha sucedido después de haber hecho los pasos anteriores. Ejecute la orden *whoami* ¿Qué usuario es actualmente?, ¿Con qué contraseña entro?, ¿Presenta esto un riesgo para la seguridad del sistema?, ¿Especifique cómo lo resolvería?
11. Lea las páginas de manual para la orden *shutdown*. Utilice este comando para apagar este sistema de varias formas distintas: con demora, reiniciando, deteniendo el sistema, forzando la ejecución de *fsck*, etc.
12. ¿Qué hacen los comandos *reboot* y *halt*? ¿Cuál es su equivalente utilizando *shutdown*?

13. Programe la ejecución de un apagado del sistema para dentro de cinco minutos. Observe la información que aparece en pantalla. Antes de que pasen los cinco minutos, cancele el apagado.

NOTA: La imagen: *grubcircle.xpm.gz*, será proporcionada por su docente.

Administración de sistemas operativos

Práctica 5: Administración de usuarios y grupos

OBJETIVOS

- Entender los procedimientos que se sigue el sistema para añadir un usuario.
- Manejar las herramientas que nos proporciona el sistema para la gestión de usuarios y grupos.
- Realizar distintas operaciones de gestión con los usuarios existentes en el sistema.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de dos sesiones de laboratorio, cada una de dos horas para un total de cuatro horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

PRÁCTICA 5
Administración de usuarios y grupos

TABLA DE CONTENIDOS:

Introducción.....	313
El archivo <i>/etc/passwd</i>	314
El archivo <i>/etc/shadow</i>	314
El archivo <i>/etc/group</i>	314
El archivo <i>/etc/gshadow</i>	315
Como añadir usuarios al sistema.....	315
Herramientas para gestionar los usuarios y grupos.....	317
adduser.....	317
groupadd.....	317
gpasswd.....	317
newgrp.....	318
chage.....	318
pwck.....	319
grpck.....	319
chsh.....	320
chfn.....	320
Supresión de usuarios o grupos.....	320
userdel.....	320
usermod.....	321
Comunicación entre administrador y usuarios.....	321
Ejercicios.....	322

Introducción

Una de las principales responsabilidades del administrador del sistema Unix es mantener las cuentas de usuarios y de grupos de usuarios. Ello incluye dar de alta nuevas cuentas, eliminar las que no se utilicen, establecer mecanismos de comunicación con los usuarios, etc. En todas las operaciones anteriores se ven implicados principalmente cuatro archivos en los que se guarda la información concerniente a los usuarios y a los grupos a los que pertenecen. Estos archivos son */etc/passwd*, */etc/shadow*, */etc/group* y */etc/gshadow*.

En esta práctica desarrollaremos habilidades para administrar las cuentas de usuarios existentes en el sistemas mediante diversos mecanismos, ya sean estos editando directamente los archivos correspondientes o haciendo uso de los comandos que nos proporciona el sistema para trabajar con las cuentas de usuarios.

El archivo */etc/passwd*

Este archivo está compuesto por una serie de líneas formadas por campos separados por dos puntos. Cada línea guarda información de un usuario y tiene un formato como el siguiente:

nombre_us:clave:us_ID:grupo_ID:coment:dir_inicio:prog_inicio

- **nombre_us:** Es el nombre de usuario o nombre de *login* que damos cada vez que iniciamos una sesión en el sistema. Es recomendable que tenga entre uno y ocho caracteres.
- **clave:** Este campo es el correspondiente a la palabra clave o clave de acceso, que está encriptada por el sistema. Cuando en dicho campo aparece una *x* indica que la palabra clave encriptada reside en el archivo */etc/shadow*.
- **us_ID:** Es el número de identificación de usuario. Para usuarios normales este identificador debe estar entre 1000 y 64999, ambos incluidos. El número cero es un identificador especial que corresponde al usuario *root*.
- **grupo_ID:** Es el número de identificación de grupo. Este número se asocia a una línea o entrada en el archivo */etc/group*.
- **coment:** Aquí aparecerá un comentario sobre el usuario, tal como su nombre completo, número de teléfono, dirección, etc.
- **dir_inicio:** Es el camino completo del directorio de inicio (*/home*) del usuario al que accederá cada vez que inicie sesión.
- **prog_inicio:** Corresponde al programa que se debe ejecutar cada vez que entre el usuario al sistema, Generalmente, este programa será el *shell* con el que queremos trabajar.

El archivo */etc/shadow*

Contiene las contraseñas encriptadas del sistema. Este archivo a diferencia de */etc/passwd* sólo puede ser leído por *root*, con el objetivo de evitar que cualquiera pueda realizar un ataque de fuerza bruta sobre el archivo que contiene las contraseñas.

Los dos campos de mayor importancia en este archivo son el primero (*nombre_us*) el cual contiene el nombre del usuario y el segundo (*clave*) que contiene la contraseña encriptada del mismo. El resto de los campos tienen que ver con la gestión de las contraseñas, por ejemplo, cuando se creó la contraseña, cuando caduca, con cuántos días de anticipación le avisamos el usuario, etc.

Si el campo *clave* dentro del archivo */etc/shadow* se encuentra vacío, la cuenta del usuario no tendrá ninguna contraseña asociada. Sin embargo, Si contiene un signo de admiración, significará que la cuenta se encuentra bloqueada.

NOTA: Existen dos algoritmos de encriptación que son ampliamente utilizados en los sistemas Unix para cifrar las contraseñas de los usuarios, dichos algoritmos son MD5 y DES. Si el campo *clave* empieza con los caracteres: *\$1*, indica que la contraseña ha sido encriptada con MD5.

El archivo */etc/group*

Este archivo está compuesto por una serie de líneas formadas por campos separados por dos puntos. Cada línea de éstas se corresponde con un grupo de usuarios y tiene un formato como el siguiente:

nombre_grupo:password:grupo_ID:lista_componentes_grupo

- **nombre_grupo:** Corresponde al nombre del grupo que está asociado con el número identificador de grupo.
- **password:** Corresponde con la contraseña del grupo. Las contraseñas son almacenadas en el archivo */etc/gshadow*.
- **grupo_ID:** Corresponde al número identificador de grupo, que debe ser igual al que aparezca en los usuarios que pertenezcan a dicho grupo en el archivo */etc/passwd*.
- **lista_componentes_grupo:** Es una lista separada por comas de los nombres de usuarios que pueden convertirse en miembros del grupo con la orden *newgrp*, no es por tanto una lista de miembros actuales del grupo.

El archivo */etc/gshadow*

Es un fichero de texto, donde cada línea tiene información de un grupo definido en */etc/group*. Nos permite guardar las contraseñas cifradas de los grupos y es sólo visible por *root*.

Debido a que las contraseñas asociadas a los grupos no son muy utilizadas, no analizaremos los campos contenidos en dicho archivo.

Como añadir usuarios al sistema

Para añadir usuarios al sistema se deben seguir, en el orden que aparecen, los siguientes pasos:

1. Copiar el actual archivo */etc/passwd* en */etc/passwd.ORIGINAL* con objeto de poder deshacer los cambios en caso de que algo falle.
2. Editar el archivo */etc/passwd* y añadir la línea o entrada correspondiente al usuario que queremos crear.

Vamos a ver a continuación una serie de normas que nos ayudarán a rellenar la línea correspondiente al nuevo usuario. En primer lugar, el nombre de acceso o nombre de conexión del usuario; es recomendable que no exceda los ocho caracteres. El campo siguiente, el correspondiente a la clave, deberá contener una *x* lo cual indicará que la contraseña será almacenada en el archivo */etc/shadow*. En el campo UID pondremos el número que identificará al nuevo usuario. Debemos elegir un identificador diferente al de cualquier otro usuario y que sea superior a 999 e inferior a 65000, ya que si no es así se producirán problemas. Para saber de forma fácil el número identificador del usuario que debemos colocar en dicho campo, podemos hacer uso del siguiente comando:

```
$ sudo cat /etc/passwd | awk -F ':' '($3>MAX && $3<65000){MAX=$3}END{print MAX+1}'
```

El campo GID lo rellenaremos con el identificador de grupo correspondiente al grupo al cual deba pertenecer el nuevo usuario. Para saber de forma fácil el número identificador de grupo que debemos colocar en dicho campo, podemos hacer uso del siguiente comando:

```
$ sudo cat /etc/passwd | awk -F ':' '($4>MAX && $4<65000){MAX=$4}END{print MAX+1}'
```

En el campo siguiente colocaremos información relacionada con la persona en cuestión: nombre completo, teléfono, dirección, etc. Seguidamente definiremos, en el siguiente campo, cuál será el directorio de arranque del nuevo usuario, dando la ruta completa de aquel (*/home/<nombre_us>*). Por último es necesario definir cuál será el programa de inicio, normalmente el *shell (/bin/bash)*.

3. Cifrar la clave del usuario para colocarla en el archivo */etc/shadow*. Esto lo podemos hacer con la siguiente orden:

```
$ mkpasswd -H MD5 <contraseña_en_texto_plano>
```

Dicho comando nos devolverá la contraseña cifrada que hemos de colocar en el archivo */etc/shadow*. Para ello debemos añadir una nueva línea o entrada correspondiente al usuario que estamos creando. Para añadir esta línea podemos utilizar como guía las que ya existen.

No debemos olvidarnos de hacer una copia (*/etc/shadow.ORIGINAL*) del archivo */etc/shadow*, con objeto de poder deshacer los cambios en caso de que algo falle.

4. Copiar el actual archivo */etc/group* en */etc/group.ORIGINAL* con objeto de poder deshacer los cambios en caso de que algo falle.
5. Añadir una línea en el archivo */etc/group* con el nombre del usuario. Debe haber una correspondencia del valor del campo **GID** de */etc/passwd* y el valor del campo **GID** de */etc/group*. Para añadir esta línea podemos utilizar como guía las que ya existen.
6. Copiar el actual archivo */etc/gshadow* en */etc/gshadow.ORIGINAL* con objeto de poder deshacer los cambios en caso de que algo falle.
7. Añadir una línea en el archivo */etc/gshadow* con el nombre del usuario. Para añadir esta línea podemos utilizar como guía las que ya existen.
8. Crear el directorio *HOME* para el nuevo usuario (el mismo que declaramos en el archivo */etc/passwd*), para ello haremos uso del siguiente comando:

```
$ sudo cp -rf /etc/skel /home/<nombre_us>
```

9. Modificamos el dueño y el grupo de los archivos ubicados en */home/<nombre_us>*, esto va a permitir que el usuario tenga permisos de hacer lo que él desee en su directorio de conexión. Para esto debemos usar los siguientes comandos:

```
$ sudo chown -R /home/<nombre_us>
```

```
$ sudo chgrp -R <nombre_us> /home/<nombre_us>
```

10. Para controlar finalmente si hemos modificado correctamente los archivos */etc/passwd*, */etc/shadow*, */etc/group* y */etc/gshadow* podemos utilizar las siguientes órdenes:

```
$ sudo pwck
```

```
$ sudo grpck
```

11. Iniciar una sesión con el nombre de usuario que acabamos de crear y comprobar que todo funciona correctamente.
12. Borrar todos los archivos *.ORIGINAL* que hemos creado con objeto de poder deshacer los cambios en caso de que algo fallara.

Herramientas para gestionar los usuarios y grupos

Para evitar llevar a cabo las labores realizadas en la sección anterior, el sistema nos proporciona un conjunto de herramientas que nos evitan modificar los archivos de administración de usuarios y grupos manualmente. Veremos a continuación algunas de ellas.

adduser

Sintaxis: `adduser usuario`

Esta orden se utiliza para dar de alta a nuevos usuarios en el sistema. Si no se proporcionan argumentos, `adduser` tomará determinados valores por defecto.

Si queremos añadir un nuevo usuario antes tenemos que definir un grupo al que pertenecerá dicho usuario. Por ejemplo, para crear el grupo de *usuarios* utilizaremos la siguiente orden:

```
$ sudo groupadd usuarios
```

Ahora ya tenemos un grupo de usuarios al que añadir un nuevo usuario:

```
$ sudo useradd -g usuarios -c "Javier Matinez" javi
```

Una vez creado el usuario debemos asignarle una contraseña utilizando la orden `passwd`.

```
$ sudo passwd javi  
Changing password for user javi  
New password:  
Retype new password:  
passwd: all authentication tokens updated successfully
```

groupadd

Sintaxis: `groupadd grupo`

Con esta orden podemos dar de alta a un nuevo grupo en el sistema. Por ejemplo, para dar de alta al grupo de usuarios de Internet llamado *usr_inet* utilizaríamos al siguiente orden:

```
$ sudo groupadd user_inet
```

gpasswd

Sintaxis: `gpasswd grupo`

El administrador del sistema es el encargado de nombrar un administrador para el grupo. Dicho administrador puede ser un usuario cualquiera del sistema. El administrador de grupo tendrá la potestad de incluir nuevos usuarios en su grupo. Sólo *root* puede establecer quién será el administrador de un grupo. Por ejemplo, para definir a *javier* como administrador del grupo *usr_inet* utilizaríamos la orden:

```
$ sudo gpasswd -A javier user_inet
```

A partir de ahora, el usuario *javier* puede añadir nuevos miembros al grupo *usr_inet*.

```
javier$ gpasswd -a usuario01 usr_inet
Adding user usuario01 to group usr_inet
```

newgrp

Sintaxis: `newgrp grupo`

Cuando se da de alta un usuario en el sistema se le asigna un grupo primario. En los ejemplos anteriores, el grupo primario para el usuario *javier* es *usuarios*. Para consultar a qué grupo primario pertenece un usuario podemos utilizar la siguiente orden:

```
javier$ id
uid=1001(javier) gid=1001(usuarios) grupos=1001(usuarios),1002(usr_inet)
```

Un usuario puede cambiarse de grupo haciendo uso de la orden *newgrp*.

```
javier$ newgrp usr_inet
javier$ id
uid=1001(javier) gid=1002(usr_inet) grupos=1001(usuarios),1002(usr_inet)
```

chage

Sintaxis: `chage -l usuario`

Con esta orden podemos manipular los tiempos máximos y mínimos en los que los usuarios deben cambiar sus contraseñas. La forma más sencilla de invocar esta orden es mediante el modificar *-l*.

```
$ sudo chage -l javier
```

Con esto se obtienen los parámetros actuales de tiempo de la cuenta del usuario *javier*.

```
Minimum:          0
Maximum:          23
Warning:          4
Inactive:         4
Last Change:      nov 06, 2007
Password Expires: nov 29, 2007
Password Inactive: dic 03, 2007
Account Expires:  dic 12, 2008
```

- **Minimum:** Indica el tiempo mínimo en días que deben transcurrir para que un usuario pueda cambiar su contraseña. Si vale cero, significa que el usuario puede cambiar su contraseña en cualquier momento. Podemos alterar este valor con la opción *-m* de la orden *chage*.
- **Maximum:** Indica el tiempo en días a partir del último cambio de cambio de la contraseña, en el que el usuario debe cambiar su contraseña. Podemos alterar este valor con la opción *-M* de la orden *chage*.
- **Warning:** Indica con cuántos días de antelación se avisará a un usuario de que su contraseña está a punto de caducar. Podemos alterar este valor con la opción *-W* de la orden *chage*.
- **Inactive:** Indica cuántos días de plazo se deja al usuario desde que caduca su contraseña hasta que la cuenta queda bloqueada. Una vez que se bloquea una cuenta el usuario no

puede acceder de nuevo hasta que el administrador la desbloquee. Podemos alterar este valor con la opción `-I` de la orden `chage`.

En el ejemplo anterior, *javier* modificó su contraseña por última vez el 6 de noviembre de 2007 (*password change*). Se estableció un tiempo máximo de duración de 23 días (*maximum*), por lo tanto la contraseña del usuario caducará el 29 de noviembre de 2007 (*password expires*). 4 días después (*inactive*), es decir, el 3 de diciembre de 2007 se procederá a bloquear la cuenta del usuario.

El administrador puede modificar cualquiera de estos parámetros. Por ejemplo, puede establecer la fecha en la que el usuario modificó por última vez una contraseña. Esto es útil para forzar que un usuario cambie su contraseña.

```
$ sudo chage -d0 javier
$ chage -I javier
Minimum:          0
Maximum:         23
Warning:         4
Inactive:        4
Last Change:     Never
Password Expires: Never
Password Inactive: Never
Account Expires: dic 12, 2008
```

La próxima vez que el usuario intente acceder se le obligará a que cambie su contraseña. Si no lo hace, no se le permitirá acceder al sistema.

pwck

Sintaxis: `pwck`

La orden `pwck` (*password check*) busca en el archivo `/etc/passwd` posibles errores de formato, así como posibles inconsistencias (usuarios duplicados, usuarios sin directorio de inicio, errores sintácticos, etc.).

Ejemplo:

```
$ sudo pwck
usuario news: el directorio /var/spool/news no existe
usuario uucp: el directorio /var/spool/uucp no existe
usuario nobody: el directorio /nonexistent no existe
pwck: sin cambios
```

grpck

Sintaxis: `grpck`

La orden `grpck` (*group check*) busca en el archivo `/etc/group` posibles errores de formato e inconsistencias avisándonos de ello.

chsh

Sintaxis: chsh

La orden *chsh* (*change shell*) puede emplearla un usuario para cambiar su intérprete de órdenes. Como sabemos, el intérprete de órdenes es el último campo de cada línea del archivo */etc/passwd*. La forma de operar de esta orden es muy similar a la orden *passwd*, con la diferencia de que lo que se modifica en este caso es el *shell* del usuario. Cuando queremos cambiar nuestro intérprete de órdenes, *chsh* visualiza el *shell* que estamos empleando y nos pide que introduzcamos uno nuevo. El nuevo intérprete de órdenes debe ser uno de los indicados en el archivo */etc/shells*, a no ser que se el propio administrador del sistema el que invoca la orden. Si el archivo */etc/shells* no existe, los únicos *shells* válidos son */bin/sh* y */bin/csh*.

chfn

Sintaxis: chfn

La orden *chfn* se utiliza para actualizar información relativa al usuario, como nombre completo, teléfono del despacho, teléfono del trabajo y teléfono de la casa, en el archivo */etc/passwd*. Cuando se nos pregunta acerca de la información anterior, se nos ofrecen unos valores por defecto encerrados entre corchetes. Este valor por defecto se acepta simplemente pulsando la tecla *Enter*. Para incluir un campo en blanco, debemos introducir la palabra *none*.

Supresión de usuarios o grupos

Para suprimir usuario definitivamente, lo único que tenemos que hacer es borrar sus entradas en los archivos */etc/passwd*, */etc/shadow*, */etc/group* y */etc/gshadow* donde aparezca el nombre de login (un usuario puede estar incluido en mas de una entrada en el archivo */etc/group*). Seguidamente podemos borrar el directorio de conexión del usuario suprimido. Para suprimir un grupo borraremos su entrada del archivo */etc/group* y */etc/gshadow*, pero siempre que ningún usuario pertenezca ya a ese grupo. Para desactivar o borrar temporalmente un usuario, esto es, no darle permiso a acceder al sistema sin borrar sus entradas en los mencionados archivos, podemos simplemente editar el archivo */etc/shadow* e introducir en el campo de la clave un !.

Por ejemplo:

```
pepe:!!$agr0ST0z$pr.4R6ESxddPit/fQy6gB/:14028:0:99999:7:::
```

Para reactivarlo, sólo tendremos que borrar el ! y dejarlo como estaba.

Para evitar editar los archivos manualmente, los sistemas Unix ponen a nuestra disposición los siguientes comandos:

userdel

Sintaxis: userdel [-r] usuario

La orden *userdel* nos permite eliminar usuarios del sistema. Por ejemplo, si queremos eliminar al usuario *javier*, tendríamos que escribir lo siguiente:

```
$ sudo userdel javier
```


A partir de este momento, el usuario eliminado ya no existe. Si además queremos eliminar también su directorio HOME, deberemos emplear la orden:

```
$ userdel -r javier
```

Es aconsejable eliminar las cuentas de usuarios que ya no se conectan al sistema, ya que éstas pueden ser agujeros en la seguridad.

usermod

Sintaxis: `usermod [opciones] usuario`

La orden `usermod` nos permite cambiar varios atributos de usuarios, entre los más utilizados tenemos:

- Desactivar una cuenta de usuario:

```
$ sudo usermod -L <usuario>
```
- Reactivar una cuenta de usuario:

```
$ sudo usermod -U <usuario>
```
- Cambiar el *shell* del usuario:

```
$ sudo usermod -s <dirección_de_nuevo_shell> <usuario>
```
- Cambiar el directorio *HOME* del usuario con todos sus contenidos:

```
$ usermod -m <nueva_ubicacion> <usuario>
```
- Cambiar el UID del usuario:

```
$ usermod -u <nuevo_UID> <usuario>
```
- Cambiar el grupo predeterminado del usuario:

```
$ usermod -g <nuevo_GID> <usuario>
```

Comunicación entre administrador y usuarios

En este punto se citarán los modos que existen para la intercomunicación del administrador con los usuarios. Consideraremos sólo aquellos mecanismos específicos. Básicamente estos modos de comunicación son la orden *wall* (*write all*) y el archivo *motd* (*message of the day*).

- **wall:** Esta utilidad del administrador envía simultánea e inmediatamente un mensaje a todos los usuarios que estén en ese momento conectados al sistema.
- **/etc/motd:** Este archivo es impreso en pantalla cada vez que un usuario inicia una sesión.

Ejercicios

1. Añada un nuevo usuario de nombre *lucas* al sistema. Este usuario debe pertenecer al grupo *users*, su directorio de arranque debe ser */home/lucas* y su programa de inicio */bin/bash*. Compruebe que *lucas* puede iniciar una sesión correctamente. A continuación desactive su cuenta y compruebe si puede o no iniciar una sesión.

NOTA: Realice el ejercicio sin hacer uso de los comandos proporcionados por el sistema.

2. Reactive la cuenta de *lucas* e iniciando una sesión como *lucas*, modifique su información personal, nombre, oficina, teléfono, etc.

NOTA: Realice el ejercicio sin hacer uso de los comandos proporcionados por el sistema.

3. Fuerce al usuario *lucas* a cambiar su contraseña la próxima vez que se conecte haciendo uso de la orden *chage*.
4. Cree un nuevo grupo denominado *documentación* y añada al usuario *lucas* a ese grupo con la orden *gpasswd*. Cree un nuevo usuario *leoncio* y añádalo también al grupo.
5. Modifique la *shell* de inicio del usuario *lucas*, mediante el comando *chsh*, para que ahora sea */bin/sh*.
6. Cree al usuario *pepe* con el comando *adduser*. Desactive su cuenta con el comando *usermod*. Reactive su cuenta con el comando *usermod* y borre al usuario pepe y sus archivos en el directorio HOME, mediante el comando *userdel*.
7. Coloque en el archivo */etc/mod* un mensaje de aviso en donde se especifique la fecha de entrega de esta práctica. Pruebe por usted mismo la correcta visualización de dicho aviso.

Administración de sistemas operativos

Práctica 6: Administración del sistema de archivos

OBJETIVOS

- Manejar las herramientas que nos proporciona el sistema para la administración del sistema de archivos.
- Aprender a establecer de forma correcta los permisos sobre archivos importantes del sistema.
- Manejar las herramientas de montaje y desmontaje de un sistema de archivos.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de dos sesiones de laboratorio, cada una de dos horas para un total de cuatro horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

PRÁCTICA 6
Administración del sistema de archivos

TABLA DE CONTENIDOS:

Introducción.....	325
Creación de enlaces con <i>ln</i>	326
<i>ln</i>	326
Uso de archivos: permisos.....	327
<i>chmod</i>	327
<i>umask</i>	328
Permisos especiales sobre archivos.....	328
Creación del sistema de archivos.....	330
<i>mkfs</i>	330
Montaje de un sistema de archivos.....	331
<i>mount</i>	332
<i>umount</i>	332
El archivo <i>/etc/fstab</i>	333
Ejercicios.....	335

Introducción

La administración del sistema de archivos es uno de los aspectos más importantes que debe tener en cuenta el administrador del sistema. Es bien sabido que cuando se instala un disco duro nuevo, a los dos días ya está medio lleno, obedeciendo a la siguiente máxima: los archivos de usuario siempre tienden a ocupar el máximo espacio posible. Para evitar esto, el administrador debe preocuparse de que cada uno de los usuarios mantenga limpio su espacio de disco (labor ardua, por otro lado). Además de eso, es necesario que el administrador sepa como añadir nuevos discos, darles formato, montar en ellos un sistema de archivos, etc. Es por ello que en esta práctica estudiaremos todas estas funciones.

Creación de enlaces con *ln*

ln

Sintaxis: *ln* archivo(s) destino

La orden *ln* (*link*) se utiliza para permitir que un mismo archivo aparezca en el sistema de archivos bajo dos nombres diferentes, pero con una única copia. Con *ln* no se hace ni se hace una copia del archivo origen, solamente se crea otro nombre de archivo que hace referencia al mismo archivo físico. Eso permite que una única copia de un archivo aparezca en varios directorios con distintos nombres. De este modo, se puede compartir información de forma cómoda. Si en un momento eliminamos alguno de los archivos que hacen referencia a la misma copia física, sólo eliminaremos el nombre, pero no la copia real. Ésta sólo será definitivamente suprimida si eliminamos todos sus vínculos (*links*). El número de enlaces de un archivo lo indica el segundo campo de la información que obtenemos con la orden: *ls -l*.

Vamos a analizar un ejemplo con objeto de dejar más claro su funcionamiento. Supongamos que tenemos un archivo, que denominamos *pss*. Usando la orden *ls -li* podemos visualizar su número de *inodo*. El número de *inodo* es un valor interno utilizado por el sistema de archivos que permite localizar toda la información relacionada con el propio archivo (tamaño, propietario, grupo, derechos de acceso, tipo de archivo, punteros a los bloques de disco, etc.).

```
$ ls -li pss
147468 pss
```

Nuestro archivo *pss* tiene un número de *inodo* igual a 147468 en el sistema de archivos. Ahora vamos a crear otro enlace a *pss* denominado *masp*. Para ello, utilizaremos la orden:

```
$ ln pss masp
```

Vamos a ver de nuevo el número de *inodo* para el archivo enlazado *masp*.

```
$ ls -li masp
147468 masp
```

Como podemos comprobar, ambos archivos tienen el mismo número de *inodo*, de manera que accediendo a *pss* o a *masp* estamos accediendo al mismo archivo físico, ya que el sistema de archivos utiliza el mismo identificador de *inodo* en ambos casos. Cualquier cambio realizado en el primero de ellos se manifestará en el segundo, y viceversa.

A este tipo de enlaces se les conoce con el nombre de enlaces fuertes o *hard links*. El problema de este tipo de enlaces es que no sirven para archivos que se encuentran en sistemas de archivos diferentes (por ejemplo, diferentes particiones del disco). Los enlaces duros tampoco son aplicables a directorios. Para solventar estos problemas podemos hacer uso de otros tipos de enlaces, denominados enlaces simbólicos o *soft links*. Un enlace simbólico tiene una funcionalidad similar a un enlace duro, pero es posible utilizarlo en archivos que se encuentren en diferentes sistemas de archivos así como enlazar directorios. Para crear enlaces simbólicos, se utiliza la orden *ln* con la opción *-s* (*soft*).

Ejemplo:

```
$ ln -s pss assp
```

De esta forma, hemos creado un enlace a *pss* apuntado por *assp*. Si ahora utilizamos la orden *ls -li* comprobaremos que ambos archivos tienen un número de *inodo* diferente:

```
$ ls -i pss assp
147469 assp 147468 pss
```

Utilizando la orden `ls -l`, podremos comprobar cómo `masp` es un enlace al primer archivo:

```
$ ls -l pss assp
lrwxrwxrwx 1 gateway gateway 3 nov 19 17:48 assp -> pss
-rw-r--r-- 2 gateway gateway 4098 nov 19 17:50 pss
```

La primera `l` incluida junto con el campo de derechos del archivo `assp` indica que este archivo es un enlace simbólico a `pss`. Los permisos de un enlace simbólico no se utilizan (aparecen siempre `lrwxrwxrwx`). En estos casos, los derechos del archivo enlace son los mismos que los del archivo destino (en nuestro ejemplo `pss`). En este caso, también tanto `pss` como `assp` hacen referencia a la misma información. Debemos tener cuidado con los enlaces simbólicos, ya que si eliminamos el archivo que actúa como destino del enlace, el archivo que lo enlazaba seguirá existiendo y apuntará a un archivo no existente. Esto es así por que el sistema, al contrario de lo que ocurría en los enlaces duros, no mantiene constancia del número de veces que un archivo se encuentra enlazado simbólicamente en el sistema de archivos.

Uso de archivos: permisos

El sistema Unix proporciona la posibilidad de proteger la información. Para ello, asocia a cada archivo una serie de derechos de acceso. En función de éstos, se determina qué es lo que cada usuario puede hacer con el archivo.

chmod

Sintaxis: `chmod modo archivo(s)`

La orden `chmod` (*change mode*) va a permitirnos modificar los permisos de un archivo. Para poder modificar estos derechos, debemos ser los propietarios del mismo. También el usuario `root` tiene la posibilidad de cambiarlos. Si no somos ni el propietario del archivo ni el administrador, `chmod` fallará. Para cambiar el modo de un archivo seguiremos estos pasos:

1. Convertir los campos de protección a dígitos binarios, poniendo un uno en el caso de que queramos activar dicho campo (`rwx`), o un cero en el caso de querer desactivarlo. Si, por ejemplo, queremos que los permisos finales del archivo sean `rwxr-xr--`, la secuencia de dígitos binarios sería: `111101100`.
2. Dividir esos dígitos binarios en tres partes de tres bits cada una: una para el usuario (propietario), otra para el grupo y una última para el resto de los usuarios (otros), de tres dígitos cada una.
3. Convertir cada grupo de tres dígitos a numeración octal.
4. Reunir los tres dígitos octal en un único número, el cual será el modo que le pasemos como argumento a `chmod`.
5. Si, por ejemplo, queremos dejar un archivo con el modo `rwxr-xr--`, lo haremos de la siguiente forma:

Modo	Usuario	Grupo	Otros
<code>rwXr-xr--</code>	<code>rwX</code>	<code>r-x</code>	<code>r--</code>
Valor binario	111	101	100
Valor octal	7	5	4

Ejemplo:

```
$ ls -l socrun
```

```
-rw-r--r-- 1 gateway gateway 4098 nov 20 13:05 socrun
```

```
$ chmod 754 socrun
```

```
-rwxr-xr-- 1 gateway gateway 4098 nov 20 13:05 socrun
```

umask

Sintaxis: `umask [máscara]`

Los permisos asignados a un archivo o a un directorio cuando son creados dependen de una variable denominada *user mask*. Podemos visualizar dicha variable dando la orden *umask* sin argumentos. El resultado son tres dígitos octales que indican, de izquierda a derecha, el valor de la máscara que determina los permisos iniciales para el propietario, para el grupo y para el resto de los usuarios. Si deseamos que por defecto nuestros archivos y directorios se creen con permisos distintos a los de la máscara actual, podremos cambiar el valor de la máscara de usuario dando la orden *umask* con el argumentos oportuno.

Permisos especiales sobre archivos

Cada archivo (ya sea ordinario, directorio o personal) contiene en su *inodo* el UID de su propietario y el *GID* de su grupo propietario, el conjunto de permisos de lectura, escritura y ejecución para el propietario, grupo y otros, además de datos adicionales concernientes al archivo. Este conjunto de permisos determina cuándo un proceso puede ejecutar una acción (lectura, escritura o ejecución) en un archivo dado. En archivos ordinarios, estas tres acciones son obvias. En directorios, la acción de escritura significa poder modificar el directorio añadiendo o borrando una entrada en el mismo, mientras que la acción de ejecución significa que pueda ser incluido en un *PATH* (por ejemplo, para utilizar *find*, o para acceder a él con la orden *cd*). En archivos especiales las acciones de lectura y escritura significan la posibilidad de poder utilizar las llamadas al sistema *read* y *write*.

Este sistema de permisos funciona de la siguiente manera:

- Si el número de identificación de usuario efectivo es 0, entonces se dan los permisos como propietario (0 es el UID efectivo del administrador del sistema).
- Si el número de identificación de usuario efectivo coincide con el número de identificación de usuario propietario del archivo marcado en su *inodo*, entonces se dan los permisos de propietario establecidos.
- Si el número de identificación de grupo efectivo coincide con el número de identificación de grupo propietario del archivo marcado en su *inodo*, entonces se dan los permisos de grupo.
- Si no se da ninguna de las tres anteriores suposiciones, se darán los permisos establecidos para otros.

NOTA: Los números de identificación de usuario efectivo y los números de identificación de grupo efectivo se usan para determinar los permisos, mientras que los números de identificación de usuario reales y los números de identificación de grupo reales se usan para saber la identidad y pertenencia a un grupo verdadera de un usuario.

Hasta ahora hemos considerado los derechos de lectura, escritura y ejecución asociados al propietario del archivo, al grupo al que pertenece el usuario y al resto de las personas. Estos derechos se representan por nueve bits. Además de estos nueve bits de derechos asociados a cada archivo, podemos considerar tres más, los bits diez, once y doce, conocidos como bit pegajoso (*sticky-bit*), bit de *set-gid* y bit de *set-uid*, respectivamente.

El bit *set-uid* es una idea relativamente simple que nos permite solucionar problemas relacionados con la protección. El hecho de que un programa tenga este bit activo implica que cuando ejecutemos dicho programa, éste tomará como identificador de usuario el identificador del propietario. Si el propietario fuese el administrador, entonces el programa se ejecutaría como si lo hubiese lanzado el propio administrador. De este modo, podemos explicarnos cómo un usuario normal puede modificar su palabra de clave cuando ello implica modificar el contenido de */etc/passwd*, que sólo tiene permiso de escritura por parte del administrador del sistema. La razón de permitir esta modificación es que el programa *passwd* que pertenece al administrador tiene el bit de *set-uid* activo, de modo que cuando ejecutamos ese programa, y sólo mientras ejecutamos ese programa, actuamos como si fuésemos el administrador. El bit de *set-uid* está activo cuando en la máscara de derechos del programa, en el campo de ejecución para el propietario, tiene activa una *s* en lugar de una *x*. Por ejemplo:

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 26616 2004-05-21 07:04 /usr/bin/passwd
```

También nosotros podemos poner el bit de *set-uid* activo en cualquiera de nuestros programas. De este modo, cuando otro usuario ejecute estos programas, tendrá los mismos derechos que el propietario. Este bit no se puede activar en los programas de *shell*. Veamos un ejemplo en el que activamos el bit de *set-uid* a un programa:

```
$ ls -l sim
-rwx-r-xr-x 1 gateway gateway 29308 ene 18 18:53 sim
```

Como vemos, el programa *sim* no tiene activo el bit comentado, para activarlo haremos uso de la orden *chmod*, indicando que deseamos activar el bit número doce (bit de *set-uid*) del siguiente modo:

```
$ chmod 4755 sim
$ ls -l sim
-rws-r-xr-x 1 gateway gateway 29308 ene 18 18:53 sim
```

Ahora, cuando cualquier usuario ejecute el programa *sim*, a todos los efectos, el programa actuará como si hubiese sido invocado por el propietario (*gateway*).

Al igual que existe un bit de *set-uid*, existe su equivalente aplicado al grupo, y se conoce como *set-gid*. La funcionalidad de este bit es completamente similar a la del bit de *set-uid*, pero en este caso se aplica al grupo. Para poner activo este bit, haremos también uso de la orden *chmod*, indicando que deseamos activar el bit número diez.

Ejemplo:

```
$ ls -l sisarch
-rwx-r-xr-x 1 gateway gateway 437428 ene 18 18:55 sisarch
$ chmod 2755 sisarch
$ ls -l sisarch
-rwx-r-xr-s 1 gateway gateway 437428 ene 18 18:55 sisarch
```

Por último, el *sticky-bit* tiene un uso especial para proteger archivos dentro de un determinado directorio. Cuando en un determinado directorio tenemos activados los derechos de escritura para un grupo de usuarios o para todos los usuarios existentes, implica que cualquiera de ellos podría borrar archivos de ese directorio, incluso aunque no le pertenezcan. Veamos un ejemplo que aclare el escenario planteado. Supongamos que el usuario *ssp* tiene un directorio denominado *publico* al cual tienen acceso todos los usuarios del sistema.

```
$ pwd
/home/ssp
$ ls -ld publico/
drwxrwxrwx 2 ssp ssp 4096 sep 21 18:00 publico/
```

Supongamos que en este directorio tenemos un archivo denominado *datos.ssp* que pertenece al usuario *ssp*. Si otro usuario accede a ese directorio, podrá borrar ese archivo, aunque no sea el propietario. Supongamos que el usuario *pepe* intenta borrarlo del modo siguiente:

```
$ id
uid=1002(pepe) gid=1002(pepe) grupos=1002(pepe)
$ pwd
/home/ssp/publico
$ ls -l datos.ssp
-rw-r--r-- 1 ssp ssp 941 sep 21 18:03 datos.ssp
$ rm datos.ssp
rm: remove write-protected file 'datos.ssp'? y
$ ls -l datos.ssp
ls: datos.ssp: No existe el fichero o el directorio
```

Como podemos apreciar, aunque *pepe* no sea el propietario del archivo, puede eliminarlo. Si queremos evitar esta posibilidad, podremos hacer uso del *sticky-bit* asociado al directorio. Activando este bit, los usuarios ya no podrán eliminar ni renombrar los archivos del directorio. Para ello bastaría que el usuario *ssp* pusiese el directorio *publico* con los siguientes atributos:

```
$ chmod 1777 publico/
```

Si ahora el usuario *pepe* intenta eliminar otro archivo, veremos qué ocurre:

```
$ ls -l datos1.ssp
-rw-r--r-- 1 ssp ssp 150 sep 21 18:07 datos1.ssp
$ rm datos1.ssp
rm: remove write-protected file 'datos1.ssp'? y
rm: cannot unlink 'datos1.ssp': Operación no permitida
```

Ahora la operación no puede llevarse a cabo, con lo que tendríamos protegidos los archivos del directorio especificado.

Creación del sistema de archivos

Los sistemas de archivos nuevos pueden crearse con la orden *mkfs*. Esta orden se encarga de dar formato al dispositivo indicado de modo que pueda albergar un sistema de archivos.

mkfs

Sintaxis: `mkfs [-vct] dispositivo [tamaño]`

mkfs construirá el nuevo sistema de archivos formateándolo. El parámetro dispositivo que aparece en la descripción de la orden se refiere al archivo de dispositivo empleado para acceder al periférico, y el tamaño indica el número de bloques que debe tener el sistema de archivos. Este formato indica estructurar el dispositivo con las partes necesarias para soportar un sistema de archivos: área de *boot*, superbloque, *inodos* y área de datos.

Esta orden admite opciones, algunas de las más comunes son las que se citan a continuación:

- **-v:** Modo verboso. Con esta opción se muestra por pantalla más información de la que se muestra habitualmente, relativa a las operaciones que se están realizando en cada momento. Esto puede ser útil para obtener información específica o para ayudar en las labores de depuración.
- **-c:** Indica que se realice una comprobación con objeto de verificar que todos los bloques son correctos.
- **-t:** Sirve para indicar el tipo de sistema de archivos que deseamos crear.

Ejemplo:

```
$ sudo mkfs /dev/fd0
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1025 (log=0)
96 inodes, 720 blocks
36 blocks (5.00%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
96 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 31 mounts
or 180 days, whichever comes first. Use tune2fs -c or -i to override.
```

Puesto que la orden *mkfs* ha sido ejecutada como *root*, la propiedad y el grupo del nuevo sistema de archivos creado es la ese usuario, por lo tanto, cuando montemos este sistema de archivos se aplicarán las reglas de acceso correspondientes al usuario *root* y a su grupo. Si queremos ceder la propiedad del sistema de archivos, por ejemplo a un usuario, deberemos hacer uso de las órdenes *chown* y *chgrp*.

Montaje de un sistema de archivos

Es muy común tener conectados a una misma máquina varios discos físicos, cada uno de ellos, probablemente, con distintas particiones (cada una descrita por su archivo de dispositivo). En cada una de estas particiones podemos tener un sistema de archivos diferente, y surge la necesidad de añadir este sistema de archivos al único disco lógico existente. Aunque tengamos distintos discos físicos, en Unix todos forman parte de un único disco lógico, al contrario que en otros sistemas en los que cada disco físico supone al menos un disco lógico.

La llamada al sistema *mount* sirve para conectar un determinado sistema de archivos a un disco lógico y la llamada *umount* sirve para el proceso inverso. Sin la existencia de estas llamadas al sistema, solamente se podría acceder a la información de los discos a través de sus archivos de dispositivo, que no sería demasiado práctico ni cómodo para el usuario final.

mount

Sintaxis: `mount [-tahvwr] [dispositivo] [dir]`

La orden *mount* sin parámetros mostrara los sistemas de archivos montados actualmente. Con los parámetros adecuados asocia el directorio *raíz* del sistema de archivos del dispositivo referenciado en dispositivo con el directorio que se encuentra en el sistema de archivos *raíz*.

Opciones:

- **-t:** Sirve para indicar el tipo de sistema de archivos que deseamos montar.
- **-a:** Monta todos los sistemas de archivos incluidos en */etc/fstab*.
- **-h:** Visualiza un mensaje de ayuda.
- **-v:** Modo verboso. Con esta opción se muestra por pantalla más información de la que se muestra habitualmente, relativa a las operaciones que se están realizando en cada momento. Esto puede ser útil para obtener información específica o para ayudar en las labores de depuración.
- **-r:** Monta el sistema de archivos en modo sólo lectura.
- **-w:** Monta el sistema de archivos en modo lectura-escritura. Éste es el modo por defecto.

Ejemplo:

```
$ sudo mount  
/dev/sda2 on / type ext2 (rw)  
none on /proc type proc (rw)  
/dev/sda1 on /dos type msdos (rw)  
/dev/fd0 on /mnt/floppy type ext2 (rw)
```

Como podemos apreciar en el ejemplo, la orden *mount* sin parámetros muestra todos los sistemas de archivos montados en ese instante. En concreto, y de izquierda a derecha, señala lo siguiente: el archivo de dispositivo correspondiente al sistema de archivos montado, el punto de montaje, el tipo de sistema de archivos y los derechos de acceso.

Lo mismo que montamos el sistema de archivos con la orden *mount*, podemos provocar su desligue lógico o desmontaje con la orden *umount*.

umount

Sintaxis: `umount dispositivo`

La orden *umount* disocia el sistema de archivos del dispositivo del sistema de archivos *raíz*. Para que se pueda desmontar, primero se debe desactivar, esto es, comprobar que no tiene ningún archivo abierto y que ningún usuario lo tenga como directorio actual de trabajo. Para comprobar qué procesos tienen abiertos archivos en un determinado sistema de archivos, podemos utilizar la orden *fuser*.

Ejemplo:

```
$ sudo umount /dev/fd0
```

El archivo */etc/fstab*

Este archivo mantiene información relativa a los sistemas de archivos existentes en el sistema. El siguiente ejemplo muestra el contenido del archivo */etc/fstab* para un sistema concreto.

```
$ cat /etc/fstab
LABEL=/ / ext3 defaults 1 1
none /dev/pts devpts gid=5,mode=620 0 0
none /proc proc defaults 0 0
none /dev/shm tmpfs defaults 0 0
/dev/hda3 swap swap defaults 0 0
/dev/cdrom /mnt/cdrom iso9660 noauto,owner,ro 0 0
/dev/fd0 /mnt/floppy auto noauto,owner 0 0
```

El delimitador de campo para este archivo es el tabulador o un espacio en blanco. Cada línea mantiene información sobre un sistema de archivos siguiendo la siguiente estructura.

fs_disp pun_montaje tipo_sis opciones freq sec_fsck

- ***fs_disp***: Indica qué dispositivo contiene el sistema de archivos. Puede ser un dispositivo físico conectado al ordenador, un dispositivo virtual, la ubicación de un sistema de archivos en res, etc.
- ***pun_montaje***: Indica en qué parte del sistema de archivos se montará el sistema de archivos en cuestión. Existen algunos valores especiales, por ejemplo *swap* indica que */dev/hda3* no tiene un punto de montaje porque se trata del archivo de intercambio del sistema.
- ***tipo_sis***: Indica qué tipo de sistema de archivos contiene el dispositivo especificado en *fs_disp*. Los valores sistemas de archivos más comunes que aquí pueden ser encontrados son:
 - ext2***: Es el sistema de archivos utilizado habitualmente.
 - ext3***: Es *ext2* con soporte transaccional (*journaling*).
 - msdos***: Sistema de archivos de MSDOS.
 - nfs***: Sistemas de archivos en red (*Network File System*).
 - iso9660***: Sistema de archivos de CD-ROM.
 - ntfs***: Sistemas de archivos utilizado por Windows NT/2K/XP/VISTA.
 - smb***: Sistemas de archivos en red *SAMBA*.
- ***opciones***: Es una lista de opciones separadas por comas. Existen un gran número de ellas que podemos utilizar para gestionar nuestros sistemas de archivos, entre ellas podemos destacar las siguientes:
 - auto/noauto***: Indica si el sistema de archivos se montará cuando se invoque la orden *mount -a*. Suele ser habitual que durante el proceso de arranque se invoque a *mount* de esta forma, con objeto de montar todos los sistemas de archivos necesarios.
 - async/noasync***: Indica si las operaciones de lectura y escritura sobre ese dispositivo deben de realizarse de forma asíncrona o no.
 - exec/noexec***: Permite o no ejecutar archivos binarios situados en el sistema de archivos en cuestión.
 - user/nouser***: Permite o no que el sistema de archivos sea montado por un usuario que no sea *root*. Si se elige *user*, el sistema aplicará por defecto *noexec*, *nosuid* y *nodev*, a menos que se especifique lo contrario.
 - nosuid***: Hace que se ignore el significado de los bits *SUID* y *SGID*.
 - ro***: Monta el sistema de archivos en modo de sólo lectura.
 - rw***: Monta el sistema de archivos en modo lectura y escritura.
 - defaults***: Es equivalente a las opciones *rw*, *suid*, *nouser*, *dev*, *exec*, *auto* y *async*.

Habitualmente no se permite que un usuario (aparte de *root*, evidentemente) pueda montar sistemas de archivos. Por ejemplo, si el usuario *jdp* quisiera montar un sistema de archivos tipo *ext2* residente en un disquete, en el directorio */mnt/floppy* intentaría la orden:

```
$ mount /dev/fd0 /mnt/floppy
mount:
```

El administrador de sistemas puede autorizar a los usuarios a montar determinados sistemas de archivos haciendo uso de la opción *user*. Editamos el archivo */etc/fstab*.

```
$ vim /etc/fstab
LABEL=/ / ext3 defaults 1 1
none /dev/pts devpts gid=5,mode=620 0 0
none /proc proc defaults 0 0
none /dev/shm tmpfs defaults 0 0
/dev/hda3 swap swap defaults 0 0
/dev/cdrom /mnt/cdrom iso9660 noauto,owner,ro 0 0
/dev/fd0 /mnt/floppy auto user 0 0
```

De esta forma autorizamos a los usuarios a montar un sistema de archivos del tipo *ext2* que se encuentre en la unidad de disco en el directorio */mnt/floppy*. Ahora, si el usuario *jdp* intenta montar dicho sistema de archivos podrá hacerlo.

```
$ mount /dev/fd0 /mnt/floppy
$ cd /mnt/floppy
$ ls -l
total 16
-rwxr-xr-x 1 jdp usuarios 13713 nov 8 15:22 miprog
-rw-r--r-- 1 jdp usuarios 38 nov 8 15:22 miprog.c
```

Ejercicios

1. Copie en su directorio de arranque un archivo cualquiera del directorio `/bin` y denomínelo `archivo1`. A continuación visualice el `archivo1` en formato largo. Haga un enlace del archivo anterior con un archivo denominado nuevo. ¿Cuántos enlaces tienen los archivos anteriores? ¿Es nuevo un archivo físico? ¿Qué ocurre si borramos el `archivo1`?
2. Vaya a su directorio de arranque, cree un subdirectorio denominado `.oculto`. Copie en este subdirectorio el archivo `/etc/hosts`. Copie el archivo `/bin/cp` en el directorio `.oculto` que acaba de crear. Vaya a su directorio de arranque, cree un subdirectorio denominado `copia`, mueva todos los archivos del directorio `.oculto` al directorio `copia`. Haga un enlace de los archivos que hay en `copia` al directorio `.oculto`. ¿Cuántos enlaces aparecen ahora por cada archivo? Borre los archivos de `copia`. ¿Cuántos enlaces aparecen ahora en los archivos de `.oculto`? Repita el proceso anterior, pero utilizando enlaces simbólicos.
3. Cree un subdirectorio en su directorio de arranque denominado `tmp`. Copie en ese subdirectorio el archivo `/etc/group` con el nombre de `grupo`. Cambie los derechos de este archivo para que los usuarios de su grupo y el resto de los usuarios puedan modificarlo.
4. ¿Qué valor deberíamos darle a la máscara de derechos para que todos los archivos se creasen con los atributos `rw-r--r--`?
5. Cree un subdirectorio en su directorio de arranque denominado `documentos`. Modifique los permisos de ese directorio para que usted pueda escribir y leer en él. Los miembros de su grupo sólo podrán acceder al directorio y leer sus contenidos, pero no escribir.
6. Modifique los permisos del directorio `documentos` para que puedan escribir en él los miembros del grupo. Compruebe que un usuario, diferente al suyo, miembro del grupo puede crear un directorio dentro de `documentos`. ¿Con que nombre de usuario y de grupo se crea ese directorio?

NOTA: En caso de que no exista otro usuario deberá crearlo y deberá añadirlo al grupo al que usted pertenece.

7. Pruebe activar el *sticky-bit* del directorio `documentos`. Si un usuario diferente al suyo y que pertenezca a otro grupo crea ahora un nuevo directorio ¿con qué nombre de grupo se crea ese nuevo directorio?
8. Determine qué sistemas de archivos hay montados en su sistema.
9. Pruebe crear un nuevo sistema de archivos en el disquete. Una vez creado, móntelo en un directorio denominado `/fd`. Pruebe acceder al sistema de archivos recién montado.
10. Desmonte el sistema de archivos que acaba de montar.
11. Modifique el archivo `/etc/fstab` para que el anterior sistema de archivos se montado de forma automática cuando se inicie el sistema.

Administración de sistemas operativos

Práctica 7: Administración de la red

OBJETIVOS

- Entender los elementos básicos sobre resoluciones de nombres de equipos y resoluciones de direcciones IP mediante el mecanismo de DNS.
- Manejar básicamente los comandos de configuración de red.
- Demostrar que se sabe configurar las interfaces de red existentes en el sistema.
- Demostrar que se sabe añadir elementos a las tablas de encaminamiento del sistema.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de una sesión de laboratorio, correspondiente a un periodo de dos horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

COMPLEMENTARIA

Tutorial y descripción técnica de TCP/IP

Dirección: <http://ditec.um.es/laso/docs/tut-tcpip/3376fm.html>

El sistema de red

Dirección: <http://www.trokotech.com/manuales/unixsec-html/node186.html>

Direcciones, enrutamiento y transporte

Dirección: http://structio.sourceforge.net/guias/AA_Linux_colegio/direcciones-enrutamiento-y-transporte.html

PRÁCTICA 7
Administración de la red

TABLA DE CONTENIDOS:

Introducción.....	339
Identificación.....	340
hostname.....	341
Resolución de nombres y direcciones.....	341
nslookup.....	341
dig.....	342
El archivo <i>/etc/resolv.conf</i>	343
Otros comandos de red.....	343
ping.....	343
ifconfig.....	344
route.....	345
traceroute.....	346
netstat.....	346
Ejercicios.....	347

Introducción

Hablar de Linux sin hablar de redes de ordenadores implicaría abordar el estudio de administración de este sistema operativo sin tocar un punto crucial en él: Las comunicaciones entre equipos informáticos. En cualquier lugar basado en estaciones de trabajo Linux es normal tener todas ellas conectadas mediante una red. Esto permite tener un mejor aprovechamiento de recursos como impresoras, información o potencia de cálculo. Esta red de interconexión puede extenderse a unos cuantos ordenadores próximos entre sí físicamente, separados a lo sumo unos cientos de metros, en cuyo caso hablamos de redes de área local o *LAN (Local Area Network)*, o bien puede extenderse a zonas más amplias, de ámbito nacional o internacional, en cuyo caso hablamos de redes de área extendida *WAN (Wide Area Network)*.

En esta práctica centraremos nuestros esfuerzos en la correcta configuración de las interfaces de red existentes en nuestro sistema, para ello haremos uso de una herramienta de emulación de redes llamada *Netkit*. Además estudiaremos todo el mecanismo básico de resoluciones de nombres a IP y viceversa. Y también analizaremos el uso de otros comandos útiles a la hora de configurar la red en nuestros sistemas Linux.

Identificación

Es necesario conocer cómo se identifica cada computador dentro de una red. Es por esto que para el caso de TCP/IP se hace uso de un número binario de 32 bits que diferencia a cada máquina conectada a la red. Como trabajar con números en formato binario resulta molesto, normalmente se utiliza una notación conocida como notación punto decimal. En este tipo de notación tenemos cuatro dígitos decimales, comprendidos entre 0 y 255, separados por puntos.

Dicho número identifica a un único ordenador dentro de la red, y es lo que se conoce normalmente como dirección IP. Obviamente, dentro de una misma red no pueden existir dos ordenadores con la misma dirección IP.

A pesar de que la notación decimal es sencilla, es preferible trabajar con nombres lógicos, tales como *dafne*, *amon*, *rigel* o *nabuco*. Si empleamos esos nombres lógicos para identificar cada una de las máquinas de la red, deberá existir algún mecanismo para traducir cada uno de los nombres a su dirección IP. Aunque existen varios métodos de traducción, el más sencillo, aunque no el más eficiente en la mayoría de las ocasiones, consiste en definir un archivo que contenga las tablas de correspondencias. Este archivo en Unix es */etc/hosts*, y su contenido podría ser similar al mostrado seguidamente:

```
$ cat /etc/hosts
# Ejemplo de archivo de hosts
# La sintaxis de cada entrada es:
# <dirección IP> <nombre oficial> <alias>
127.0.0.1          localhost          localhost.localdomain
172.19.16.4       cardhu.unaleon.edu.ni cardhu
193.146.9.131    ra.unaleon.edu.ni  ra
```

Este archivo, como podemos observar, contiene una lista de direcciones IP, un nombre de ordenador, un alias y posiblemente algún comentario por cada línea.

Cuando Internet era pequeña, la solución anterior era factible. Cada sistema podía tener en su archivo */etc/hosts* el listado de todas las máquinas accesibles. Actualmente, debido a que existen demasiados ordenadores en Internet, la solución anterior es poco útil.

La solución adoptada para solventar el problema anterior consiste en emplear bases de datos distribuidas donde se almacenan las correspondencias entre nombres de máquina y dirección IP. Estas bases de datos son manipuladas y mantenidas por los servidores de nombres. Por razones de efectividad y flexibilidad, en vez de emplear un único servidor de nombres centralizado se emplean varios. La razón es que actualmente existen demasiadas instituciones conectadas a Internet, con lo que es poco práctico avisar a un servidor central cada vez que realizamos un cambio en nuestra propia red. Así pues, el manejo de nombres se relega a cada institución. Los servidores de nombres forman una estructura de árbol correspondiente a la estructura de instituciones. Los propios nombres de las máquinas siguen una estructura similar. Un nombre típico de ordenador podría ser *ftp*. En el caso anterior, el ordenador presentado es un servidor de *ftp* perteneciente a la Universidad Nacional Autónoma de Nicaragua. En el caso anterior, el nombre del ordenador es *ftp*. El segundo campo *unanleon* identifica a la Universidad Nacional Autónoma de Nicaragua, el tercer campo identifica que es una institución educativa y el cuarto y último campo es *ni* el cual hace referencia a Nicaragua. Del modo anterior, cualquier ordenador del mundo queda caracterizado. Al mecanismo anterior se le conoce como organización por dominios. A la terminología que se utiliza para referirnos a un nombre de dominio se la conoce como *FQDN (Fully Qualified Domain Name)*. Esta terminología suele ser la más adecuada, ya que nos permite obtener información del dominio con sólo saber su nombre.

Existen órdenes en Unix que nos permiten conocer tanto el nombre del ordenador al que estamos conectados como el dominio al cual pertenece. Si estamos trabajando en un sistema Unix y queremos saber su nombre, tendremos que emplear la orden *hostname* que se muestra a continuación.

hostname

Sintaxis: `hostname`

Ejemplo:

```
$ hostname  
apollo
```

El *hostname* es el nombre que identifica a nuestro ordenador en la red. En el ejemplo anterior, el nombre es *apollo*.

Para saber el nombre de nuestro dominio, tenemos que emplear la orden `hostname` junto con la opción `-d`

Ejemplo:

```
$ hostname -d  
it.unanleon.edu.ni
```

En el ejemplo anterior, el dominio asociado a la máquina *apollo*, a la que estamos conectados, es *it.unanleon.edu.ni*.

Resolución de nombres y direcciones

Anteriormente indicamos la necesidad de referirnos a las distintas máquinas por su nombre lógico y no por su dirección IP. Aunque esta traducción se puede hacer a escala local, lo normal es emplear los servicios de lo que se conoce como servidores de nombres. Estos servidores, como ya hemos indicado, son máquinas especializadas en realizar esta labor de traducción. Normalmente, dichos servidores forman parte de una base de datos distribuida, lo cual permite que la base de datos sea más fiable que una centralizada y, además, cada máquina no necesita almacenar toda la información. En caso de que un servidor de nombres no conozca la IP de una determinada máquina, puede preguntárselo a otro servidor. De este modo se establece una jerarquía en árbol que permite que todo funcione perfectamente.

Si necesitamos conocer la dirección IP o la dirección lógica de algún ordenador en el mundo, podemos utilizar el programa *nslookup*, cuya funcionalidad y sintaxis se muestra a continuación.

nslookup

Sintaxis: `nslookup [máquina]`

La orden *nslookup* se emplea para determinar la dirección IP de un ordenador del cual sólo conocemos su nombre lógico, o bien para conocer su nombre lógico sabiendo su dirección IP. El programa tiene dos modos de trabajo, el interactivo y el no interactivo. En nuestro caso sólo estudiaremos el interactivo. Para entrar en modo interactivo, no pasaremos ninguna opción, y se utilizará como servidor de nombres el que esté configurado por defecto.

Ejemplo:

\$ nslookup

Default Server: dulcinea.unanleon.edu.ni

Address: 130.206.82.7

> **209.124.106.186** (*Quiero saber el nombre de la máquina cuya dirección IP es la indicada*)

Server: dulcinea.unanleon.edu.ni

Address: 130.206.82.7

Name: pintur.intur.gob.ni (Respuesta)

Address: 209.124.106.186

> **laprensa.com.ni** (*Quiero saber la dirección IP de la máquina cuyo nombre es el indicado*)

Server: dulcinea.unanleon.edu.ni

Address: 130.206.82.7

Name: laprensa.com.ni (Respuesta)

Address: 208.96.128.164

> **exit**

dig

Sintaxis: `dig @s_dns dominio t_cons c_cons +opt_con -dig_opt`

Existe una tendencia a ir eliminando la utilidad *nslookup* en favor de los programas *dig* y *host*. La orden *dig* utiliza los siguientes parámetros:

- **@s_dns**: Es el servidor DNS al que queremos enviar la consulta. Este campo es opcional. Si lo omitimos, *dig* utilizara el servidor de nombres del sistema (*/etc/resolv.conf*).
- **dominio**: Es el nombre del dominio en el que estamos interesados.
- **t_cons**: Es el tipo de información que estamos buscando, por ejemplo:
 - a**: Dirección de red.
 - any**: Toda la información que exista sobre el dominio.
 - mx**: Servidores de correo para el dominio.
 - ns**: Servidores de nombres para el dominio.
 - soa**: Información administrativa sobre el dominio, por ejemplo, quién es el encargado de su gestión.
 - hinfo**: Información sobre la máquina, por ejemplo qué sistema operativo ejecuta.
- **c_cons**: Clase de consulta realizada.
- **+opt_con**: Opciones de la consulta para enviar al servidor.
- **-opt_dig**: opciones de la consulta para el programa *dig*.

La forma más sencilla de utilizar este programa es cuando preguntamos por la dirección de red de una determinada máquina, por ejemplo vamos a averiguar la dirección de red de *www.unanleon.edu.ni*.

\$ dig www.unanleon.edu.ni

```

; <<>> DiG 9.3.4 <<>> www.unanleon.edu.ni
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31921
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
www.unanleon.edu.ni      IN      A

;; ANSWER SECTION:
www.unanleon.edu.ni.    84477 IN      A      192.107.104.10

;; Query time: 41 msec
;; SERVER: 130.206.82.7#53(130.206.82.7)
;; WHEN: Wed Jun 4 11:18:29 2008
;; MSG SIZE rcvd: 53

```

El archivo */etc/resolv.conf*

Anteriormente vimos las órdenes *nslookup*, *hostname* y *dig*, las cuales son empleadas para realizar la conversión entre nombres lógicos de ordenadores y direcciones IP, y viceversa. Esta orden comprueba si los nombres o las direcciones que deseamos traducir se encuentran en el archivo */etc/hosts*, pero si no es así, no habrá más remedio que preguntar a un servidor externo que nos informe de la correspondencia entre nombre lógico y dirección IP. A estos servidores se les denomina servidores de nombres o DNS (*Domain Name Server*). El modo de indicar cual debe ser el servidor de nombres de nuestra máquina se establece configurando adecuadamente el archivo */etc/resolv.conf*. Este archivo contiene cuál es nuestro nombre de dominio y cuáles son nuestros servidores de nombres. Se puede especificar un servidor de nombres principal y otros secundarios. A continuación se muestra un ejemplo del contenido de este archivo:

```

$ cat /etc/resolv.conf
domain unanleon.edu.ni
nameserver 130.206.82.7
nameserver 130.206.1.2

```

La palabra reservada *domain* se emplea para especificar el nombre de nuestro dominio. La palabra reservada *nameserver* se emplea para especificar cuál es la dirección IP de nuestro servidor de nombres o DNS. Se pueden especificar hasta tres servidores de nombres y éstos serán consultados en el orden en que aparecen en el archivo */etc/resolv.conf*.

Otros comandos de red**ping**

Sintaxis: ping ordenador

La orden *ping* puede utilizarse para determinar si un ordenador está vivo en ese momento. Si el ordenador está vivo, contestará a *ping* por cada mensaje que reciba. *ping* sacará estadísticas de tiempo de respuesta de cada uno de los paquetes enviados al destino. Para finalizar el envío de paquetes, pulsaremos *Ctrl+c*.

Ejemplo:

```
$ ping www.google.com.ni
```

ifconfig

Sintaxis: `ifconfig interfaz [-net|-host] IPaddr [opciones]`

La orden *ifconfig* se utiliza para iniciar las interfaces de red o para mostrar información sobre las mismas. Si se invoca sin argumentos nos mostrará el estado de todas las interfaces de red que el núcleo conoce. Las opciones *-net* y *-host* se emplean para que la dirección *IPaddr* sea tratada como una dirección de red o como la dirección IP del propio ordenador, respectivamente. El argumento *interfaz* se utiliza para identificar la interfaz de red que deseamos configurar o simplemente de la que deseamos obtener información.

Las opciones más comunes que suelen emplearse son:

- **up:** Con esta opción se activa la interfaz indicada.
- **down:** Sirve para desactivar la interfaz indicada.
- **netmask <mask>:** Se utiliza para definir la máscara de red.
- **broadcast <addr>:** Se utiliza para definir la dirección de difusión. Esta dirección será empleada cuando queramos que todos los ordenadores de nuestra red reciban el mismo mensaje simultáneamente.

Para configurar una determinada interfaz de red, por ejemplo una tarjeta *Ethernet*, es necesario que el núcleo reconozca dicha interfaz. Si no es así, nunca podremos poner en marcha los servicios de red.

Suponiendo que nuestra interfaz de red se denomina *eth0*, con la siguiente orden la configuraremos para que nuestra dirección IP sea 172.29.16.5, nuestra máscara de red 255.255.255.0 y la dirección de *boradcast* 172.29.16.255.

```
$ sudo ifconfig eth0 172.29.16.5 netmask 255.255.255.0 broadcast 172.29.16.255 up
```

Si ahora utilizamos *ifconfig* sin argumentos, nos mostrará la siguiente información.

\$ ifconfig

```
eth0  Link encap:Ethernet HWaddr 00:E0:B8:EB:08:32
      inet dirección:172.29.16.5 Bcast:172.29.16.255 Máscara:255.255.255.0
      dirección inet6: fe80::2e0:b8ff:feeb:832/64 Alcance:Vínculo
      ARRIBA BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:354435 errors:0 dropped:0 overruns:0 frame:0
      TX packets:111918 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:161996285 (154.4 MiB) TX bytes:9213833 (8.7 MiB)
      Interrupción:20
lo    Link encap:Bucle local
      inet dirección:127.0.0.1 Máscara:255.0.0.0
      dirección inet6: ::1/128 Alcance:Anfitrión
      ARRIBA LOOPBACK RUNNING MTU:16436 Metric:1
      RX packets:133 errors:0 dropped:0 overruns:0 frame:0
      TX packets:133 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:12909 (12.6 KiB) TX bytes:12909 (12.6 KiB)
```


La interfaz *lo* es empleada para realizar pruebas en bucle local. Como podemos observar tiene asignada como dirección IP la 127.0.0.1.

route

Sintaxis: `route [add|del] [default] [-net|-host]
add [gw gateway] [metric n]`

Esta orden se emplea para definir cómo deben encaminarse aquellos paquetes que no van dirigidos a ningún ordenador incluido en nuestra propia red. De este modo, el núcleo puede establecer estrategias como la siguiente: “para enviar un paquete a la red externa X, utilice como pasarela la dirección del sistema Y”. Se pueden establecer diferentes modos de encaminamiento dependiendo de las redes destino a las que se envían los paquetes y también asignar un coste a cada una de las rutas. Siempre es necesario definir una ruta por la que se deben enviar aquellos mensajes que no coinciden con ninguna de las tablas de encaminamiento definidas en el núcleo, esta ruta es la que se conoce como ruta por defecto. En la mayoría de los casos no es necesario establecer una tabla de encaminamiento completa, basta con definir el encaminador (*router*) por defecto o la ruta por defecto. Lo anterior sería equivalente a decirle al núcleo: “cualquier mensaje que vaya fuera de nuestra red debe enviarse a la dirección Z”. Z sería la dirección de nuestro *router* por defecto.

Si a *route* no se le especifica ninguna opción, únicamente visualizará la tabla de encaminamiento actual. Las opciones *add* y *del* se emplean para agregar o borrar las rutas especificadas, respectivamente. Seguidamente se resume la funcionalidad del resto de opciones:

- **default:** Esta opción sólo la emplearemos para definir (*add*) el encaminador por defecto.
- **-net:** Sirve para tratar la dirección indicada como una dirección de red.
- **-host:** Sirve para tratar la dirección indicada como una dirección IP de un ordenador.
- **addr:** Es la dirección destino de la nueva ruta. Puede ser una dirección IP o una red.
- **gw gateway:** Con ello indicamos el encaminador que debe emplearse para el destino especificado.
- **metric n:** Indica el coste asociado a la ruta especificada. Estos costes podemos utilizarlos para determinar cuáles son los caminos óptimos para enviar los mensajes.

En el siguiente ejemplo definimos cual es el encaminador por defecto en nuestro sistema:

```
$ sudo add route default gw 172.29.16.1
```

Si a continuación invocamos a *route* sin argumentos, nos informará sobre las tablas de encaminamiento actuales:

```
$ route
Kernel IP routing table
Destination  Gateway      Genmask      Flags  Metric  Ref  Use  Iface
172.29.16.0  *            255.255.25.0 U      0       0    1    eth0
127.0.0.0    *            255.0.0.0    U      0       0    1    lo
default      172.29.16.1  0.0.0.0      UG     0       0    0    eth0
```

Un modo de saber la trayectoria que sigue un determinado paquete hasta llegar a su destino, así como los tiempos empleados cada vez que pasa de un sistema a otro, consiste en emplear el programa *traceroute* que describimos a continuación.

traceroute

Sintaxis: `traceoure destino`

Esta orden admite multitud de opciones que deben consultarse en el manual (*man traceroute*) para poder obtener el máximo partido. El único parámetro obligatorio es *destino*, que identifica al ordenador con el que vamos a comunicarnos.

Ejemplo:

```
$ traceroute 193.145.14.30
traceroute to 193.145.14.30 (193.145.14.30), 30 hops max, 40 byte packets
 2 193.146.56.129 (193.146.56.129) 2190.296 ms 2540.476 ms
 3 rcm-cr1-uah.redimadrid.madrimasd.org (193.145.14.30) 2380.086 ms 2220.268 ms
 2860.607 ms
```

En el ejemplo anterior se ven implicados 2 *gateways*, el número 3 es el ordenador destino. Aparecen también los tiempos empleados para transferir paquetes entre los diferentes encaminadores. Esta información puede utilizarse para ver dónde se encuentran los cuellos de botella en la transmisión y evitarlos si es posible.

netstat

Sintaxis: `netstat [-acirvn]`

La orden *netstat* se emplea para comprobar cuál es el estado global de la red TCP/IP. Si se invoca sin argumentos, mostrará las conexiones de red activas en el sistema. La orden soporta muchas opciones, algunas de las cuales se resumen a continuación:

- **-a:** Muestra información sobre todas las conexiones.
- **-c:** Muestra de forma continuada el estado de la red actualizándose a intervalos de un segundo. Esto se repetirá hasta que la orden sea interrumpida (*Ctrl+c*).
- **-i:** Muestra estadísticas de los dispositivos de red.
- **-r:** Muestra la tabla de encaminamiento del núcleo.
- **-n:** Hace que *netstat* imprima las direcciones IP en notación punto decimal en vez de usar los nombres simbólicos de las máquinas o las redes.
- **-v:** Nos informa sobre la versión de *netstat*.

Ejercicios

1. Averigüe el nombre de su máquina y el dominio al que pertenece, en caso de no pertenecer a ninguno, modifique el o los archivos necesarios para que pertenezca al dominio `unanleon.edu.ni`.
2. Determine la dirección IP que le corresponde al ordenador `www.bcn.gob.ni`. Determine también la dirección lógica (FQDN) que le corresponde a la siguiente dirección IP: `200.85.166.34`.
3. Comprueba cual es la configuración de red de su sistema.
4. ¿Cómo puede dar de baja a su interfaz de red?, ¿Qué ocurre si da de baja a su interfaz de red?
5. Visualice la tabla de encaminamiento empleada por el núcleo del sistema.
6. Cambie el servidor de nombres de su sistema y ejecute la orden `nslookup`. Configure adecuadamente la resolución de nombres para que opere lo más rápido posible.
7. Debido a la falta de equipos y a la escases de recursos que existen en los laboratorios, para el desarrollo de este inciso vamos a hacer uso de una herramienta de emulación de redes llamada *Netkit* (instalada en todas las máquinas del laboratorio).

Netkit es un sistema para emular redes de ordenadores. Es importante especificar que utilizando *Netkit* podremos emular redes y no simularlas:

- **Simulación de sistemas:** Reproducir el rendimiento de un sistema real (latencia, pérdida de paquetes, etc.).
- **Emulación de redes:** Reproducir las funcionalidades de un sistema real (configuración, arquitectura, protocolos, etc.) teniendo en cuenta las limitaciones en cuanto al rendimiento.

Netkit es un conjunto de comandos que permiten configurar y conectar fácilmente máquinas virtuales y un sistema de ficheros que contiene las herramientas necesarias para llevar a cabo la configuración en las máquinas virtuales.

Algunos comandos útiles de *Netkit*:

- **vlist:** Muestra información acerca de las máquinas virtuales que están actualmente en ejecución.
- **vstart <nombre_máquina>:** Inicia una nueva máquina virtual.
- **vcrash <nombre_máquina>:** Finaliza la ejecución de una máquina virtual que le pasemos como parámetro.
- **netgui.sh:** Arranca el entorno gráfico de emulación de *Netkit*.

Netkitgui es una interfaz gráfica que nos permite introducir los comandos de arranque y configuración necesarios para *Netkit* de una forma sencilla.

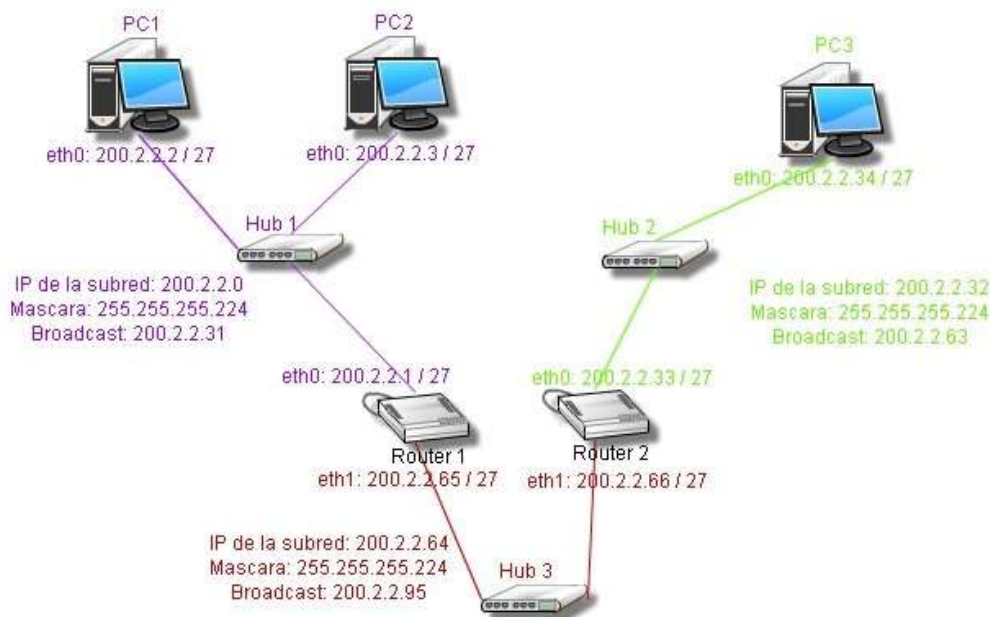
Arrancamos *netkitgui* (mediante el comando `netgui.sh`) y se nos muestran las herramientas de edición de red. Estas herramientas nos permiten crear los siguientes elementos:


- Terminales.
- Routers.
- Hubs.

- Conexión entre cualquiera de los elementos anteriores.

Una vez dibujados los elementos que queremos emular, arrancaremos la configuración que nos abrirá las correspondientes consolas de configuración de los elementos que hayamos dibujado inicialmente.

Una vez que hemos entendido la forma de funcionamiento básico de *Netkit* estamos listos para realizar el ejercicio. Este ejercicio se basa en configurar todas las interfaces en cada uno de los equipos dibujados en la figura siguiente:



Para ello debemos dibujar los elementos tal cual aparecen en la figura (tres *terminales*, tres *hubs* y dos *routers*) junto con sus respectivos nombres. Una vez que hemos dibujado todos los elementos estamos listos para configurarlos adecuadamente y que todos sean alcanzables desde cualquier punto de la red. Para arrancar la configuración dibujada debemos seleccionar el botón . Una vez que se hayan terminado de cargar todos los sistemas debemos utilizar los comandos aprendidos en las secciones anteriores para:

- Que todos los PC's puedan ser alcanzados desde cualquier lugar de la red. Para ello debemos configurar primeramente las interfaces *eth0* en cada uno de los PC's, luego debemos configurar las tablas de encaminamiento en cada uno de los *routers* para que estos puedan hacer llegar los paquetes a cualquiera de las subredes de la figura.

Para comprobar el correcto funcionamiento debemos hacer *ping* desde cada una de las máquinas hacia cualquier otra ubicada en cualquier punto de la red. Y por último debemos hacer un *traceroute* desde la PC1 hasta la PC3.

Administración de sistemas operativos

Práctica 8: Copias de seguridad

OBJETIVOS

- Aprender las distintas técnicas para respaldar y recuperar archivos.
- Aprender a utilizar los comandos básicos para realizar copias de seguridad del sistema.

TEMPORIZACIÓN

El plazo de realización de esta práctica será de una sesión de laboratorio, correspondiente a dos horas.

BIBLIOGRAFÍA

BÁSICA

UNIX y LINUX. Guía práctica, 3ª edición

Autor: Sebastián Sánchez Prieto y Óscar García Población

Editorial: Ra-Ma

Edición: 2005

COMPLEMENTARIA

Capítulo 12. Copias de seguridad (Backups).

Dirección: <http://www.ibiblio.org/pub/Linux/docs/linux-doc-project/system-admin-guide/translations/es/html/ch12.html>

PRÁCTICA 8
Copias de seguridad

TABLA DE CONTENIDOS:

Introducción.....	351
find.....	352
Órdenes para realizar copias de seguridad.....	354
cpio.....	354
tar.....	355
Ejercicios.....	357

Introducción

Una de las principales responsabilidades del administrador del sistema es preservar los datos almacenados en el sistema, planificando y realizando copias de seguridad de forma regular. La cantidad de información que debemos copiar y cada cuánto tiempo debemos hacerlo, dependerá de la utilización que se haga del sistema. Para ver la cantidad de información que debemos copiar frecuentemente, nos fijaremos en aquellas áreas que son modificadas continuamente. Los volcados totales del sistema se harán con mucha menos frecuencia. Sin una buena política de copias de seguridad se pueden llegar a perder datos valiosos. Un sistema es tanto más seguro cuanto más frecuentemente se hagan las copias de seguridad. En esta práctica estudiaremos las órdenes más empleadas para realizar copias de seguridad.

Antes de empezar a ver las órdenes que nos ayudarán a realizar las copias de seguridad es necesario que entendamos el funcionamiento de una orden complementaria que nos va a permitir obtener las ubicaciones de los archivos sobre los cuales queremos realizar las copias de seguridad, la orden *find*.

find

Sintaxis: *find* camino expresión

La orden *find* es una de las más potentes de Unix, pero también una de las que tienen una sintaxis más compleja. Esta orden se utiliza para examinar toda la estructura de directorios, o bien la parte que le indiquemos, buscando los archivos que cumplan los criterios señalados en la línea de órdenes. Una vez localizados, podemos hacer que ejecute distintas acciones sobre ellos. El campo *expresión* sirve para indicar los criterios de selección de los archivos y la acción que queremos aplicarles al encontrarlos.

Veamos con un ejemplo como podemos buscar un determinado archivo dentro de la estructura de directorios.

Ejemplo:

```
$ sudo find / -name ifconfig  
/sbin/ifconfig
```

La opción *-name* indica a *find* que únicamente se busquen los archivos cuyo nombre se especifica a continuación, y la opción *-print* indica a *find* que visualice el nombre del archivo por pantalla una vez hallado (en muchos sistemas el modificador *-print* se toma como valor por defecto).

Existen muchas más opciones para la orden *find*, entre estas tenemos:

- **-user:** Con esta opción, *find* seleccionará los archivos que pertenezcan al usuario que se indique a continuación de *-user*.
- **-group:** *find* seleccionará los archivos pertenecientes al grupo indicado a continuación.
- **-mtime <n>:** *find* seleccionará los archivos modificados hace <n> días.
- **-mtime -<n>:** *find* seleccionará los archivos modificados en los últimos <n> días.
- **-mtime +<n>:** *find* seleccionará los archivos modificados hace más de <n> días.
- **-size -<m>:** *find* seleccionará los archivos cuyo tamaño es menor que <m> bloques.
- **-size +<m>:** *find* seleccionará los archivos cuyo tamaño es mayor que <m> bloques.
- **-type <x>:** *find* seleccionará los archivos del tipo <x>, donde <x> puede ser:
 - b:** Archivo especial de modo bloque.
 - c:** Archivo especial de modo carácter.
 - d:** Directorio.
 - p:** Tubería con nombre (FIFO).
 - f:** Archivo regular.
 - l:** Enlace simbólico.
 - s:** Conector de comunicaciones (*socket*).

Todos los operadores anteriores pueden ser negados con el carácter *!* seguido de un espacio en blanco. En este caso, *find* buscará todos los archivos que no cumplan la especificación indicada.

Se pueden especificar simultáneamente varias opciones, en cuyo caso se seleccionarán los archivos que cumplan todas ellas (operación *and*). Si dichas opciones las conectamos con el operador *-o* (operación *or*), *find* seleccionará los archivos que cumplan cualquiera de ellas.

Como por ejemplo imaginemos que queremos visualizar todos los archivos que cuelguen de nuestro directorio *HOME* cuyo tamaño sea mayor de 1500 bloques y que hayan sido modificados en los últimos cinco días.

```
$ find $HOME -size +1500 -mtime -5 -print
/home/gateway/.galeon/mozilla/galeón/Cache/24C15940d01
/home/gateway/.galeon/history.xml
/home/gateway/Libro/Libro.ps
/home/gateway/Libro/Libro.dvi
```

También podemos indicar a *find* que ejecute una orden determinada y la aplique a los archivos que encuentre. Para construir la orden que queremos ejecutar con cada archivo que encuentre *find* contamos con la expresión *{}* que se sustituye por el nombre del archivo encontrado. Debemos además concluir la orden con el carácter *“;”*. Hay que tener en cuenta que muchos intérpretes de órdenes (*bash* por ejemplo) consideran a *“;”* como un carácter especial, por lo tanto será necesario colocar una secuencia de escape para evitar dicha interpretación, es decir *\;*

Vamos a poner seguidamente unos ejemplos de usos típicos de *find*. En el primero encontraremos todos los archivos que cuelguen de */usr/bin* que sean enlaces simbólicos a otros archivos, haciendo que la información presentada en pantalla sea de la forma:

```
Archivo: [nombre_archivo] es enlace simbólico
```

Para construir ese literal podemos emplear la orden *echo* de la siguiente forma:

```
echo Archivo: {} es enlace simbólico \;
```

Así, la orden *find* completa sería:

```
$ sudo find /usr/bin -type l -exec echo Archivo: {} es enlace simbólico \;
Archivo: /usr/bin/X11 es un enlace simbólico
Archivo: /usr/bin/sg es un enlace simbólico
Archivo: /usr/bin/captoinfo es un enlace simbólico
Archivo: /usr/bin/infotocap es un enlace simbólico
Archivo: /usr/bin/reset es un enlace simbólico
Archivo: /usr/bin/awk es un enlace simbólico
...
```

En un segundo ejemplo borraremos de nuestro directorio *raíz* todos los archivos que hayan sido modificados en los últimos dos días y cuyo nombre termine en *.tmp*. Para ello deberíamos emplear una orden como la siguiente:

```
$ sudo find / -mtime -2 -type f -name *.tmp -exec rm -i {} \;
```

Órdenes para realizar copias de seguridad

cpio

Sintaxis: `cpio -o [cvx]`
`cpio -i [dcruvmfx] [modelos]`
`cpio -p [dvmrx] directorio`

Descripción:

- **cpio -o**: Lee la entrada estándar para obtener una lista de archivos y los copia en la salida, junto al estado de dichos archivos. Se utilizará para la realización de copias de seguridad.
- **cpio -i**: Extrae archivos de la entrada estándar si coinciden con los modelos que pueden aparecer como argumentos. Por defecto, estos modelos corresponderán al “*”, el cual referencia a todos los archivos. Los archivos extraídos serán condicionalmente creados y copiados en el directorio actual según las opciones que lleve la orden. Se utilizará para la recuperación de la información volcada en el dispositivo.
- **cpio -p**: Lee la entrada estándar para obtener una lista de archivos que son creados y copiados, según las opciones que lleve la orden, en el directorio que aparece como argumento obligatorio.

Opciones:

- **-c**: Escribe o lee cabeceras de información en caracteres ASCII.
- **-d**: Crea directorios si es necesario.
- **-f**: Sólo copia los archivos que no se adaptan al patrón especificado.
- **-m**: Mantiene la fecha de modificación de los archivos cuando se crean archivos.
- **-r**: Renombra los archivos interactivamente.
- **-u**: Copia incondicionalmente, aunque el archivo ya exista.
- **-v**: Imprime una lista de los nombres de los archivos.

Ejemplos:

```
$ find $HOME -type f -name *.c | cpio -o > BackupArchivosC.cpio
25 blocks
$ file BackupArchivosC.cpio
BackupArchivosC.cpio: cpio archive
```

Con la orden anterior volcaremos hacia el archivo *BackupArchivosC.cpio* los archivos fuente en lenguaje C que existen en el directorio actual. Al preguntar a continuación por el tipo de archivo, con la orden *file*, podemos apreciar que el archivo es de tipo *cpio*.

```
$ cpio -i < BackupArchivosC.cpio
25 blocks
```

En el ejemplo anterior restauraremos los archivos originales almacenados en el archivo *BackupArchivosC.cpio*.

```
$ sudo find /etc | cpio -pdm /seguridad
3150 blocks
```

Con la orden anterior copiaremos todos los archivos situados en el directorio */etc* y los de sus posibles subdirectorios en el directorio */seguridad*.

tar

Sintaxis: `tar [opciones] [archivo(s)]`

La orden *tar* (*tape archive*) se utilizó en sus comienzos para guardar o recuperar archivos en una cinta magnética. *tar* puede ser utilizado para almacenar o recuperar información de cualquier archivo o dispositivo genérico como disquetes, disco regrabables o archivos ordinarios.

Si alguno de los archivos que deseamos realizar copia es un directorio, *tar* recorrerá todo el directorio y posibles subdirectorios para recoger toda la información contenida en los mismos.

Las opciones no necesitan ser precedidas del guión (salvo algunas que son especiales). Las acciones *tar* están controladas por medio de una clave, la cual es una cadena de caracteres formada por una letra, llamada de función, seguida de una o más letras llamadas modificadores de función. Las letras de función pueden ser las siguientes:

- **c**: Crea un nuevo archivo escribiendo desde el principio del archivo, destruyendo lo que había.
- **r**: Añade archivo al final de archivo.
- **t**: Lista los nombres de todos los archivos del archivo.
- **u**: Sólo añade los archivos que son más nuevos que los contenidos en la copia realizada con *tar*.
- **x**: Se utiliza para extraer del archivo *tar* los archivos indicados.

Los modificadores de función son los siguientes:

- **f <arch>**: Los archivos serán almacenados o extraídos del archivo *<arch>*. *<arch>* normalmente es un archivo de dispositivo correspondiente a una cinta o un disco flexible, aunque puede ser cualquier archivo. Si *<arch>* es el carácter -, se utilizará como archivo de dispositivo la entrada estándar o la salida estándar.
- **l**: Mostrará mensajes acerca de los enlaces simbólicos que no se encuentren.
- **m**: Provoca que no se actualice la fecha de modificación (en caso de extracción) escrita en el archivo.
- **p**: Hace que *archivo* obtenga los modos originales, así como el propietario y grupos escritos en el archivo.
- **v**: Normalmente, *tar* trabaja silenciosamente sin mostrar mensajes. En modo verboso *tar* escribe el nombre de cada archivo procesado con la letra de función que rige la acción.
- **w**: Fuerza a *tar* a pedir la confirmación de la acción a realizar con cada archivo.
- **L**: Sigue los enlaces simbólicos.
- **Z**: La información es comprimida o descomprimida con el programa *compress*.
- **z**: La información es comprimida o descomprimida con el programa *gzip*.

Ejemplos:

```
$ tar cvfz /dev/fd0 *  
  latex8.bst  
  latex8.dvi  
  latex8.log  
  latex8.sty  
  latex8.tex
```

En el ejemplo anterior hemos llevado al disquete todos los archivos del directorio actual comprimidos con *gzip*. Hay que tener cuidado con la orden anterior, ya que toda la posible

información contenida originalmente en el disquete se perderá. Además, la única forma de volver a leer la información sería utilizando *tar*, el posible formato original del disquete se perderá también.

Ahora veamos cómo podemos determinar la información contenida en el disquete:

```
$ tar tvzf /dev/fd0
```

```
-rw-rw-r-- gateway/users 19466 2007-06-20 13:09:33 latex8.bst  
-rw-rw-r-- gateway/users 10836 2007-06-20 13:09:33 latex8.dvi  
-rw-rw-r-- gateway/users 8954 2007-06-20 13:09:33 latex8.log  
-rw-rw-r-- gateway/users 4653 2007-06-20 13:09:33 latex8.sty  
-rw-rw-r-- gateway/users 9466 2007-06-20 13:09:33 latex8.tex
```

Para recuperar la información almacenada en el disquete tendríamos que utilizar la orden:

```
$ tar xvzf /dev/fd0
```

```
latex8.bst  
latex8.dvi  
latex8.log  
latex8.sty  
latex8.tex
```

La orden *tar* puede emplearse también (y suele utilizarse mucho) para enviar la información a un archivo ordinario al que por costumbre se le suele poner la extensión *.tar*.

Ejemplo:

```
$ tar cvf euromicro.tar *
```

```
latex8.bst  
latex8.dvi  
latex8.log  
latex8.sty  
latex8.tex
```

```
$ ls -l *.tar
```

```
-rw-r--r-- 1 gateway gateway 122880 jun 19 18:35 euromicro.tar
```

Ejercicios

1. ¿Qué orden utilizaría para realizar una copia de seguridad completa del sistema utilizando el comando *tar*?
2. Usando el comando *tar*, ¿Cómo se muestra en pantalla el contenido de la copia de seguridad generada en el ejercicio anterior?
3. Usando el comando *tar*, ¿Qué orden utilizaría para recuperar todos los ficheros existentes dentro de la copia de seguridad realizada en el ejercicio 1?, ¿Desde que ubicación dentro del sistema de archivos ejecutaría dicha orden?
4. Usando el comando *tar* para realizar la copia de seguridad. Indique cuál sería la orden necesaria para realizar una copia de seguridad de todos los archivos modificados las últimas veinticuatro horas.
5. ¿Se puede utilizar el comando *tar* para copiar una estructura de directorios, preservando los permisos, dueños, grupos, fechas y enlaces?, ¿Con que comando?
6. ¿Qué orden utilizaría para realizar una copia de seguridad completa del sistema utilizando el comando *cpio*?
7. Usando el comando *cpio*, ¿Cómo se muestra en pantalla el contenido de la copia de seguridad generada en el ejercicio anterior?
8. Usando el comando *cpio*, ¿Qué orden utilizaría para recuperar todos los ficheros existentes dentro de la copia de seguridad realizada en el ejercicio 6?, ¿Desde que ubicación dentro del sistema de archivos ejecutaría dicha orden?
9. Usando el comando *cpio* para realizar la copia de seguridad. Indique cuál sería la orden necesaria para realizar una copia de seguridad de todos los archivos modificados durante la semana pasada.
10. Investigue ¿Cómo se podría realizar una copia de seguridad completa de nuestro directorio raíz utilizando el comando *dd*?. Indique cuál sería el comando necesario para comprimir dicha copia de seguridad haciendo uso de una única sentencia. ¿Cuál es la desventaja de realizar copias de seguridad haciendo uso del comando *dd*?
11. Investigue, ¿Cuál sería el comando que tendríamos que utilizar para realizar una copia de seguridad de nuestro MBR?, ¿Cómo restauraríamos dicha copia de seguridad de nuestro MBR?
12. Investigue, ¿Cuál sería el comando que tendríamos que teclear para realizar una copia de seguridad de nuestras particiones?, ¿Cómo restauraríamos dicha copia de seguridad de nuestra particiones?

XI. BIBLIOGRAFÍA

Bibliografía Básica:

- Matthias Kalle Dalheimer y Matt Welsh, “Guía de referencia y aprendizaje LINUX Segunda edición actualizada y ampliada “, Editorial Anaya Multimedia, 2006.
- Dee-Ann LeBlanc, “Administración de sistemas LINUX La biblia”. Editorial ANAYA MULTIMEDIA, 2001.
- Jack Tackett Jr. y David Gunter, “Linux Tercera Edición, Edición Especial”, Editorial Prentice Hall, 1998.
- M Carling, Stephen Degler, James Dennis, “Administración de Sistemas Linux Guía Avanzada”. Editorial Prentice Hall, 2000.
- Sebastián Sánchez Prieto, “Sistemas operativos, textos universitarios, segunda edición”, Editorial universidad de Alcalá.
- Sebastián Sánchez Prieto, Óscar García Población, “UNIX y LINUX Guía Práctica Tercera edición”. Editorial Ra-Ma, 2005.

Referencias de Internet:

Las referencias a Internet se encuentran al inicio de cada uno de los capítulos del temario y al inicio de cada una de las prácticas de laboratorio.

