Curso de Linux Básico

por Fernando Ferrer

Curso de Linux Básico

por Fernando Ferrer

Tabla de contenidos

1

El sistema de nombres de dominio (DNS)

Tabla de contenidos

1.1. Funcionamiento de DNS

El *Domain Name System* (DNS) o Sistema de Nombres de Dominio permite a los usuarios de una red TCP/IP utilizar nombres jerárquicos y descriptivos para localizar fácilmente ordenadores (*hosts*) y otros recursos en dicha red, evitando de esta manera tener que recordar la dirección IP de cada ordenador al que se desea acceder. En esencia, DNS es una base de datos distribuida que contiene asociaciones de nombres simbólicos (de hosts) a direcciones IP. El hecho de que sea distribuida permite delegar el control sobre diferentes segmentos de la base de datos a distintas organizaciones, pero siempre de forma que los datos de cada segmento están disponibles en toda la red, a través de un esquema cliente-servidor.

Los programas denominados servidores de nombres (*name servers*) constituyen la parte servidora del esquema cliente-servidor. Los servidores de nombres contienen información sobre algunos segmentos de la base de datos y los ponen a disposición de los clientes, llamados solucionadores o *resolvers*.

1.1.1. El Espacio de Nombres de Dominio

La base de datos distribuida de DNS está indexada por nombres de dominio. Cada nombre de dominio es esencialmente una trayectoria en un árbol invertido denominado *espacio de nombres de dominio*. La estructura jerárquica del árbol es similar a la estructura del sistema de ficheros UNIX. El árbol tiene una única raíz en el nivel superior llamada raíz (*root*). Cada nodo del árbol puede ramificarse en cualquier número de nodos de nivel inferior. La profundidad del árbol está limitada a 127 niveles.

Cada nodo en el árbol se identifica mediante una etiqueta no nula que puede contener hasta 63 caracteres, excepto el nodo raíz, identificado mediante una etiqueta nula. El nombre de dominio completo

de cualquier nodo está formado por la secuencia de etiquetas que forman la trayectoria desde dicho nodo hasta la raíz, separando cada etiqueta de la siguiente mediante un punto. De esta forma, el nombre del nodo especifica de forma unívoca su localización en la jerarquía. A este nombre de dominio completo o absoluto se le conoce como *nombre de dominio completamente cualificado* o *Fully Qualified Domain Name* (FQDN). Al ser nula la etiqueta que identifica el nodo raíz, el FQDN de cualquier nodo del árbol siempre acaba con un punto. La única restricción que se impone en el árbol de nombres es que los nodos hijos del mismo padre tengan etiquetas diferentes.

En el esquema jerárquico de nombres DNS, se denomina *dominio* a cualquier subárbol del espacio de nombres de dominio. De esta forma, cada dominio puede contener, a su vez, otros dominios. Generalmente, los hosts están representados por las hojas del árbol, aunque es posible nombrar a un host con una etiqueta correspondiente a un nodo intermedio del árbol (en este caso, tendríamos un dominio y un nodo que se llaman igual).

La información sobre los nombres de dominio DNS se guarda mediante los denominados *registros de recursos* en los servidores DNS de la red. Concretamente, cada servidor DNS contiene los registros de recursos necesarios para responder a las consultas sobre la parte del espacio de nombres en la que tiene autoridad.

1.1.2. El Espacio de Nombres de Dominio en Internet

El estándar DNS no impone muchas reglas sobre las etiquetas de los nombres de dominio, ni tampoco asocia un significado determinado a las etiquetas de un determinado nivel del espacio de nombres. Cuando manejamos una parte de este espacio, podemos decidir el significado y la sintaxis de nuestros nombres de dominio. Sin embargo, en el espacio de nombres Internet existente, se ha impuesto una estructura de nombres bien definida, especialmente en los dominios de primer nivel.

Los dominios originales de primer nivel dividían originalmente el espacio de nombres de Internet en siete dominios: com, edu, gov, mil, net, org, e int. Posteriormente, para acomodar el crecimiento y la internacionalización de Internet, se reservaron nuevos dominios de primer nivel que hacían referencia a países individuales.

Actualmente, los dominios originales se denominan *dominios de primer nivel genéricos* y han surgido nuevos nombres que se ajustan a los tiempos que corren.

1.1.3. Delegación

Es importante resaltar que el objetivo principal del diseño del sistema de nombres de dominio fue su administración descentralizada. Este objetivo se consigue a través de la *delegación*. La delegación de dominios funciona de forma parecida a la delegación de tareas en una organización. Un responsable de proyecto divide el proyecto en pequeñas tareas y asigna (delega) la responsabilidad de las mismas a diferentes empleados.

De la misma forma, una organización que administra un dominio puede dividirla en subdominios. Cada subdominio puede ser delegado a diferentes organizaciones, lo cual implica que esa organización será responsable de mantener los datos (registros de recursos) de ese subdominio. Esa organización puede libremente cambiar los datos e incluso volver a dividir el dominio delegado en subdominios y delegarlos. El dominio padre solamente contiene enlaces a los responsables del subdominio delegado, de forma que pueda hacer referencia a ellos cuando se le planteen consultas sobre nombres en dicho subdominio delegado.

Realmente, la subdivisión de un dominio en subdominios y la delegación de dichos subdominios son cosas distintas. En primer lugar, un dominio que tenga capacidad de autogestión (autoridad), siempre puede decidir subdividirse en diferentes subdominios, manteniendo él en principio la autoridad sobre todos ellos. Posteriormente, la organización que gestiona el dominio puede decidir además delegar la autoridad de algunos (o todos) sus subdominios en otras organizaciones. La delegación es una acción

que siempre decide el dominio padre, y éste puede revocarla cuando desee, volviendo a retomar la autoridad sobre el subdominio que había delegado.

1.1.4. Servidores de nombres y zonas

Como se ha dicho anteriormente, los programas que almacenan información sobre el espacio de nombres de dominio se denominan servidores de nombres. En virtud de la delegación mencionada anteriormente, cada servidor de nombres posee generalmente información completa sobre una *parte contigua* del espacio de nombres (generalmente un dominio, potencialmente dividido en subdominios). Dicha parte del espacio se denomina *zona*, y se dice que el servidor de nombres tiene *autoridad* sobre ella. En realidad, un mismo servidor de nombres puede tener autoridad sobre múltiples zonas, y obtiene la información que describe la zona (los registros de recursos) o bien de un fichero local o bien de otro servidor de nombres.

Entender la diferencia entre una zona y un dominio es importante. Todos los dominios de primer nivel, y la mayoría de dominios de segundo nivel, se dividen en unidades más pequeñas y manejables gracias a la delegación. Estas unidades se denominan zonas y contienen una serie de registros almacenados en un servidor. Sin embargo, las zonas no son dominios. Un dominio es un subárbol del espacio de nombres, mientras que una zona es una parte del espacio de nombres DNS que se almacena generalmente en un fichero y que puede contener informació sobre múltiples dominios.

DNS define dos tipos de servidores de nombres que mantienen informacion sobre el espacio de nombres: primarios (maestros) y secundarios (esclavos). Un servidor de nombres primario para una zona lee los datos de la zona desde un fichero que él mantiene. Un servidor de nombres secundario para una zona obtiene los datos de la zona desde otro servidor de nombres que es autoritario para la zona, llamado servidor maestro. Normalmente el servidor maestro es el servidor primario de la zona, pero esto no es un requisito ya que un servidor secundario puede cargar los datos desde otro secundario.

Cuando un servidor de nombres secundario se inicia, éste se pone en contacto con su servidor maestro y, si es necesario, inicia una transferencia de zona, es decir, una actualización de su información sobre la zona (ver Sección 1.2.4). Además, periódicamente el servidor secundario contacta con el servidor maestro para ver si los datos de zona han cambiado. Tanto el servidor primario como el secundario poseen autoridad sobre la zona. Definir servidores secundarios proporciona tolerancia a errores y reduce la carga en el servidor primario de la zona.

1.1.5. Resolución de nombres

Los clientes DNS utilizan bibliotecas llamadas solucionadores (*resolvers*) que efectúan las consultas DNS a los servidores en nombre del cliente.

Los servidores de nombres son los expertos en obtener información del espacio de nombres de dominio. Es decir, no solamente responden los datos referentes a las zonas sobre los que tienen autoridad, sino que pueden también buscar información a través del espacio de nombres de dominio para encontrar datos sobre los que no son autoritarios. A este proceso se le denomina *resolución de nombres*. Por ese motivo, existen servidores de nombres que no mantienen información sobre ninguna zona, y únicamente sirven para responder consultas de los clientes (*resolvers*) sobre cualquier dominio. Este tipo de servidores DNS se denomina *cache only*.

Ya que el espacio de nombres está estructurado como un árbol invertido, un servidor de nombres necesita únicamente los nombres de dominio y las direcciones de los servidores de nombres raíz para encontrar cualquier punto en el árbol. Los servidores raíz conocen dónde se encuentran los servidores de nombres con autoridad para los dominios de primer nivel. De hecho, la mayoría de servidores raíz son autoritarios para los dominios de primer nivel genéricos.

Cuando se solicita una consulta a cualquier nombre de dominio, los servidores raíz pueden al menos proporcionar los nombres y direcciones de los servidores de nombres autoritarios para el dominio de

primer nivel al que pertenece el nombre de dominio buscado. Y los servidores de nombres de primer nivel pueden proporcionar la lista de servidores de nombres autoritarios para el dominio de segundo nivel al que pertenece el nombre de dominio buscado. De esta forma, cada servidor de nombres consultado va proporcionando la información más próxima a la respuesta buscada, o proporciona la propia respuesta.

Como conclusión hay que resaltar la importancia que tienen los servidores de nombres raíz en el proceso de resolución. Por esta razón, el sistema de nombres de dominio proporciona mecanismos de caché para ayudar a reducir la carga que supondría el proceso de resolución sobre los servidores raíz. Si todos los servidores raíz de Internet fallaran por un largo período de tiempo, toda la resolución en Internet fallaría. Para protegerse, Internet posee 13 servidores de nombres raíz repartidos por diferentes partes de la Red.

1.2. Configuración de DNS

Los estándares de DNS no especifican la estructura de datos interna en que deben almacenarse los registros de recursos (registros de la base de datos DNS), y por tanto existen varias implementaciones que son diferentes en este sentido. Por regla general, los servidores guardan la información sobre las zonas en ficheros en texto plano sin formato. Los nombres de los archivos son arbitrarios y se especifican en la configuración del servidor DNS.

Por ejemplo, en la implementación habitual de DNS en el mundo UNIX, BIND (*Berkeley Intenet Name Domain*), se utiliza los nombres de archivo siguientes para almacenar los registros de cada zona:

- `Db.dominio`: zona de resolución directa.
- `Db.direccion`: zona de resolución inversa.
- `Db.cache`: sugerencias de servidores raíz.
- `Db.127.0.0.1`: resolución inversa de bucle cerrado.

Por el contrario, la configuración predeterminada del servidor DNS de Microsoft Windows 2000 no utiliza los mismos nombres de archivo que BIND, sino que usa la nomenclatura `nombre_zona.dns`. Por otra parte, en Windows 2000, la base de datos DNS puede integrarse con la base de datos de Active Directory, en cuyo caso dicha información participa de los mismos mecanismos de almacenamiento y replicación que el resto de información contenida en el Directorio Activo.

1.2.1. Registros de Recursos (RR)

Para resolver nombres, los servidores consultan sus zonas. Las zonas contienen *registros de recursos* que constituyen la información de recursos asociada al dominio DNS. Por ejemplo, ciertos registros de recursos asignan nombres descriptivos a direcciones IP.

El formato de cada registro de recursos es el siguiente:

```
Propietario     TTL     Clase     Tipo     RDATA
```

donde:

- **Propietario**: nombre de host o del dominio DNS al que pertenece este recurso. Puede contener un nombre de host/dominio (completamente cualificado o no), el símbolo "@" (que representa el

nombre de la zona que se está describiendo) o una cadena vacía (en cuyo caso equivale al propietario del registro de recursos anterior).

- **TTL**: (*Time To Live*) Tiempo de vida, generalmente expresado en segundos, que un servidor DNS o un resolver debe guardar en caché esta entrada antes de descartarla. Este campo es opcional. También se puede expresar mediante letras indicando días (d), horas (h), minutos (m) y segundos (s). Por ejemplo: "2h30m".

- **Clase**: define la familia de protocolos en uso. Suele ser siempre "`IN`", que representa Internet.

- **Tipo**: identifica el tipo de registro.

- **RDATA**: los datos del registro de recursos.

A continuación se describen los principales tipos de registros de recursos: SOA, NS, A, PTR, CNAME, MX y SRV.

1.2.1.1. Registro de Recurso SOA

Cada zona contiene un registro de recursos denominado Inicio de Autoridad o SOA (*Start Of Authority*) al comienzo de la zona. Los registros SOA incluyen los siguientes campos (sólo se incluyen los que poseen un significado específico para el tipo de registro):

- **Propietario**: nombre de dominio de la zona.

- **Tipo**: "`SOA`".

- **Persona responsable**: contiene la dirección de correo electrónico del responsable de la zona. En esta dirección de correo, se utiliza un punto en el lugar del símbolo "@".

- **Número de serie**: muestra el número de versión de la zona, es decir, un número que sirve de referencia a los servidores secundarios de la zona para saber cuándo deben proceder a una actualización de su base de datos de la zona (o *transferencia de zona*). Cuando el número de serie del servidor secundario sea *menor* que el número del maestro, esto significa que el maestro ha cambiado la zona, y por tanto el secundario debe solicitar al maestro una transferencia de zona. Por tanto, este número debe ser incrementado (manualmente) por el administrador de la zona cada vez que realiza un cambio en algún registro de la zona (en el servidor maestro).

- **Actualización**: muestra cada cuánto tiempo un servidor secundario debe ponerse en contacto con el maestro para comprobar si ha habido cambios en la zona.

- **Reintentos**: define el tiempo que el servidor secundario, después de enviar una solicitud de transferencia de zona, espera para obtener una respuesta del servidor maestro antes de volverlo a intentar.

- **Caducidad**: define el tiempo que el servidor secundario de la zona, después de la transferencia de zona anterior, responderá a las consultas de la zona antes de descartar la suya propia como no válida.

- **TTL mínimo**: este campo especifica el tiempo de validez (o de vida) de las respuestas "negativas" que realiza el servidor. Una respuesta negativa significa que el servidor contesta que un registro no existe en la zona.

  Hasta la versión 8.2 de BIND, este campo establecía el tiempo de vida por defecto de todos los registros de la zona que no tuvieran un campo TTL específico. A partir de esta versión, esto último se consigue con una *directiva* que debe situarse al principio del fichero de la zona. Esta directiva

se especifica así:

```
$TTL tiempo
```

Por ejemplo, un tiempo de vida por defecto de 30 minutos se establecería así:

```
$TTL 30m
```

Un ejemplo de registro SOA sería el siguiente:

```
admon.com.  IN  pc0100.admon.com  hostmaster.admon.com.
                (
                  1      ; número de serie
                  3600   ; actualización 1 hora
                  600    ; reintentar 10 minutos
                  86400  ; caducar 1 día
                  60     ; TTL 1 minuto
                )
```

1.2.1.2. Registro de Recurso NS

El registro de recursos NS (*Name Server*) indica los servidores de nombres autorizados para la zona. Cada zona debe contener registros indicando tanto los servidores principales como los secundarios. Por tanto, cada zona debe contener, como mínimo, un registro NS.

Por otra parte, estos registros también se utilizan para indicar quiénes son los servidores de nombres con autoridad en subdominios delegados, por lo que la zona contendrá al menos un registro NS por cada subdominio que haya delegado.

Ejemplos de registros NS serían los siguientes:

```
admon.com.           IN   NS    pc0100.admon.com.
valencia.admon.com.  IN   NS    pc0102.valencia.admon.com.
```

1.2.1.3. Registro de Recurso A

El tipo de registro de recursos A (*Address*) asigna un nombre de dominio completamente cualificado (FQDN) a una dirección IP, para que los clientes puedan solicitar la dirección IP de un nombre de host dado.

Un ejemplo de registro A que asignaría la dirección IP 158.42.178.1 al nombre de dominio `pc0101.valencia.admon.com.`, sería el siguiente:

```
pc0101.valencia.admon.com.    IN   A       158.42.178.1
```

1.2.1.4. Registro de Recurso PTR

El registro de recursos PTR (*PoinTeR*) o puntero, realiza la acción contraria al registro de tipo A, es decir, asigna un nombre de dominio completamente cualificado a una dirección IP. Este tipo de recursos se utilizan en la denominada *resolución inversa*, descrita en Sección 1.1.4.

Un ejemplo de registro PTR que asignaría el nombre `pc0101.valencia.admon.com.` a la dirección IP 158.42.178.1 sería el siguiente:

```
1.178.42.158.in-addr.arpa.  IN   PTR  pc0101.admon.valencia.com.
```

1.2.1.5. Registro de Recurso CNAME

El registro de nombre canónico (CNAME, *Canonical NAME*) crea un alias (un sinónimo) para el nombre de dominio especificado.

Un ejemplo de registro CNAME que asignaría el alias `controlador` al nombre de dominio `pc0102.valencia.admon.com`, sería el siguiente:

```
controlador.valencia.admon.com.
                    IN      CNAME   pc0101.valencia.admon.com.
```

1.2.1.6. Registro de Recurso MX

El registro de recurso de intercambio de correo (MX, *Mail eXchange*) especifica un servidor de intercambio de correo para un nombre de dominio. Puesto que un mismo dominio puede contener diferentes servidores de correo, el registro MX puede indicar un valor numérico que permite especificar el orden en que los clientes deben intentar contactar con dichos servidores de correo.

Un ejemplo de registro de recurso MX que define al servidor `pc0100` como el servidor de correo del dominio `admon.com`, sería el siguiente:

```
admon.com.      IN      MX      0       pc0100.admon.com.
```

1.2.1.7. Registro de Recurso SRV

Con registros MX se puede especificar varios servidores de correo en un dominio DNS. De esta forma, cuando un proveedor de servicio de envío de correo necesite enviar correo electrónico a un host en el dominio, podrá encontrar la ubicación de un servidor de intercambio de correo. Sin embargo, esta no es la forma de resolver los servidores que proporcionan otros servicios de red como WWW o FTP.

Los registros de recurso de servicio (SRV, *SeRVice*) permiten especificar de forma genérica la ubicación de los servidores para un servicio, protocolo y dominio DNS determinados.

El formato de un registro SRV es el siguiente:

```
servicio.protocolo.nombre  TTL  clase  SRV
                           prioridad  peso  puerto  destino
```

donde:

- El campo `servicio` especifica el nombre de servicio: http, telnet, etc.
- El campo `protocolo` especifica el protocolo utilizado: TCP o UDP.
- `nombre` define el nombre de dominio al que hace referencia el registro de recurso SRV.

- Los campos `TTL` y `clase` ha sido definidos anteriormente.
- `prioridad` especifica el orden en que los clientes se pondrán en contacto con los servidores: los clientes intentarán ponerse en contacto primero con el host que tenga el valor de prioridad más bajo, luego con el siguiente y así sucesivamente.
- `peso`: es un mecanismo de equilibrio de carga.
- `puerto`: muestra el puerto del servicio en el host.
- `destino`: muestra el nombre de dominio completo para la máquina compatible con ese servicio.

Un ejemplo de registros SRV para los servidores Web del dominio `admon.com.`, sería:

```
http.tcp.admon.com.  IN  SRV   0  0  80  www1.admon.com.
http.tcp.admon.com.  IN  SRV  10  0  80  www2.admon.com.
```

1.2.2. Definición de la delegación

Para que una zona especifique que uno de sus subdominios está delegado en una zona diferente, es necesario agregar un *registro de delegación* y, generalmente, el denominado "registro de pegado" (*glue record*). El registro de delegación es un registro NS en la zona principal (padre) que define el servidor de nombres autorizado para la zona delegada. El registro de pegado es un registro tipo A para el servidor de nombres autorizado para la zona delegada, y es necesario cuando el servidor de nombres autorizado para la zona delegada también es un miembro de ese dominio (delegado).

Por ejemplo, si la zona `admon.com` deseara delegar la autoridad a su subdominio `valencia.admon.com`, se deberían agregar los siguientes registros al archivo de configuración correspondiente de la zona `admon.com`:

```
valencia.admon.com.         IN  NS  pc0102.valencia.admon.com.
pc0102.valencia.admon.com.  IN  A   158.42.178.2
```

1.2.3. Tipos de zonas

Aunque distintas implementaciones de DNS difieren en cómo configurar las zonas, generalmente existe un fichero que indica sobre qué zonas tiene autoridad el servidor, indicando para cada una el fichero que contiene la información de dicha zona (si el servidor es primario para la zona), o la dirección del servidor maestro a quien preguntar por ella (si es secundario).

En general, existen tres tipos distintos de zonas: zonas de búsqueda directa, zonas de búsqueda inversa y zonas de "sugerencia raíz". Un servidor DNS puede tener autoridad sobre varias zonas directas e inversas, y necesita poseer información sobre las "sugerencias raíz" si desea responder a sus clientes sobre registros de zonas sobre las que no posee autoridad. A continuacion se describe cada tipo brevemente.

1.2.3.1. Zona de búsqueda directa

Las zonas de búsqueda directa contienen la información necesaria para resolver nombres en el dominio DNS. Deben incluir, al menos, registros SOA y NS, y pueden incluir cualquier otro tipo de registros de recurso, excepto el registro de recursos PTR.

1.2.3.2. Zona de búsqueda inversa

Las zonas de búsqueda inversa contienen información necesaria para realizar las búsquedas inversas. La mayor parte de las consultas proporcionan un nombre y solicitan la dirección IP que corresponde a ese nombre. Este tipo de consulta es el descrito en la zona de resolución directa.

Pero existen ocasiones en que un cliente ya tiene la dirección IP de un equipo y desea determinar el nombre DNS de ese equipo. Esto es importante para los programas que implementan la seguridad basándose en el FQDN que se conecta y también se utiliza para la solución de problemas de red TCP/IP.

Si el único medio de resolver una búsqueda inversa es realizar una búsqueda detallada de todos los dominios en el espacio de nombres DNS, la búsqueda de consulta inversa sería demasiado exhaustiva como para realizarla de forma práctica.

Para solucionar este problema se creó un dominio DNS especial para realizar búsquedas "inversas", denominado `in-addr.arpa.`. Este dominio utiliza un orden inverso de números en la notación decimal de las direcciones IP. Con esta disposición se puede delegar la autoridad de miembros inferiores del dominio `in-addr.arpa.` a las distintas organizaciones, a medida que se les asigna identificadores de red de clase A, B o C.

1.2.3.3. Sugerencias de los servidores del Dominio Raíz

El archivo de "sugerencias raíz" (*root hint*), denominado también archivo de sugerencias de caché, contiene la información de host necesaria para resolver nombres fuera de los dominios en los que el servidor posee autoridad. En concreto, este archivo contiene los nombres y las direcciones IP de los servidores DNS del dominio punto (`.`) o raíz.

1.2.4. Transferencias de zona

En aquellas zonas en las que existen diferentes servidores de nombres con autoridad (uno principal o maestro y uno o varios secundarios o esclavos), cada vez que se realizan cambios en la zona del servidor maestro, estos cambios deben replicarse a todos los servidores secundarios de esa zona. Esta acción se lleva a cabo mediante un mecanismo denominado transferencia de zona. Existen dos tipos de transferencia de zonas: completa e incremental.

1.2.4.1. Transferencia completa de zona

En una transferencia completa de zona, el servidor maestro para una zona transmite toda la base de datos de zona al servidor secundario para esa zona.

Los servidores secundarios siguen los siguientes pasos a la hora de realizar una transferencia de zona:

1. El servidor secundario para la zona espera el tiempo especificado en el campo Actualizar del registro SOA y luego le pregunta al servidor maestro por su registro SOA.

2. El servidor maestro responde con su registro SOA.

3. El servidor secundario para la zona compara el número de serie devuelto con su propio número y si este es mayor que el suyo, solicita una transferencia de zona completa.

4. El servidor maestro envía la base de datos de la zona completa al servidor secundario.

Si el servidor maestro no responde, el servidor secundario lo seguirá intentando después del intervalo especificado en el campo Reintentos del registro SOA. Si todavía no hay respuesta después del intervalo que se especifica en el campo `Caduca` desde la última transferencia de zona, este descarta su zo-

na.

1.2.4.2. Transferencia incremental de zona

Las transferencias completas de zona pueden consumir gran ancho de banda de la red. Para poder solucionar este problema se define la transferencia incremental de zona, en la cual sólo debe transferirse la parte modificada de una zona.

La transferencia incremental de zona funciona de forma muy similar a la transferencia completa. En este caso, el servidor secundario para la zona comprueba el número de serie del registro SOA del maestro con el suyo, para determinar si debe iniciar una transferencia de zona, la cual en este caso sería incremental (sólo de los cambios realizados).

1.2.4.3. Notificación DNS

Con este proceso se pretende que el servidor maestro para la zona notifique los cambios a ciertos servidores secundarios y de esta manera los secundarios podrán comprobar si necesitan iniciar una transferencia de zona. De esta forma se mejora la coherencia de los datos mantenida por todos los servidores secundarios.

1.2.5. Actualizaciones dinámicas

Originalmente, DNS se diseñó para que solamente admitiera cambios estáticos. De esta forma, sólo el administrador del sistema DNS podía agregar, quitar o modificar los registros de recursos, realizando cambios manuales sobre los ficheros de configuración correspondientes.

El sistema de actualizaciones dinámicas, permite que el servidor principal para la zona pueda configurarse de forma que acepte actualizaciones de recursos enviadas desde otros equipos (habitualmente, sus clientes DNS). Este es el sistema preferido en el caso de Windows 2000, aunque muchos administradores de DNS lo desaconsejan por razones de seguridad.

Por ejemplo, el servidor maestro puede admitir (e incluir en su configuración) actualizaciones de registros A y PTR de las estaciones de trabajo de su dominio, que le envían esa información cuando arrancan. También sería posible recibir estas actualizaciones de un servidor DHCP, una vez ha proporcionado la configuración IP a un cliente.

2

Dominios en Linux

Tabla de contenidos

2.1. Introducción

En el mundo Linux no existe un concepto de dominio tan elaborado como en el mundo de Windows 2000. Sin embargo, se consigue un efecto similar al activar un servicio en una máquina Linux (que actuaría como "servidor" de cuentas y grupos) y otro servicio que permite la exportación de directorios a máquinas remotas. En concreto, dichos servicios se denominan *LDAP* y *NFS*, respectivamente. Ambos son explicados en este capítulo y en el siguiente, respectivamente.

2.2. Concepto de Dominio

Desde el punto de vista de la administración de sistemas, suele denominarse *dominio* a un conjunto de equipos interconectados que comparten información administrativa (usuarios, grupos, contraseñas, etc.) centralizada. Ello requiere fundamentalmente la disponibilidad de (al menos) un ordenador que almacene físicamente dicha información y que la comunique al resto cuando sea necesario, típicamente mediante un esquema cliente-servidor. Por ejemplo, cuando un usuario desea iniciar una conexión interactiva en cualquiera de los ordenadores (clientes) del dominio, dicho ordenador deberá validar las credenciales del usuario en el servidor, y obtener de éste todos los datos necesarios para poder crear el contexto inicial de trabajo para el usuario.

En Windows 2000, la implementación del concepto de dominio se realiza mediante el denominado *Directorio Activo*, un servicio de directorio basado en diferentes estándares como LDAP (Lightweight Directory Access Protocol}) y DNS (*Domain Name System*). En el mundo Unix, los dominios solían implementarse mediante el famoso *Network Information System* (NIS), del que existían múltiples variantes. Sin embargo, la integración de servicios de directorio en Unix ha posibilitado la incorporación de esta tecnología, mucho más potente y escalable que NIS, en la implementación de dominios.

Este capítulo describe cómo una implementación libre del protocolo LDAP para Unix, denominada *OpenLDAP* (`www.openldap.org`), puede utilizarse para implementar dominios en Fedora Linux.

2.3. Servicios de Directorio y LDAP

En el contexto de las redes de ordenadores, se denomina *directorio* a una base de datos especializada que almacena información sobre los recursos, u "objetos", presentes en la red (tales como usuarios, ordenadores, impresoras, etc.) y que pone dicha información a disposición de los usuarios de la red. Por este motivo, esta base de datos suele estar optimizada para operaciones de búsqueda, filtrado y lectura más que para operaciones de inserción o transacciones complejas. Existen diferentes estándares que especifican servicios de directorio, siendo el demominado *X.500* tal vez el más conocido.

El estándar X.500 define de forma nativa un protocolo de acceso denominado DAP (*Directory Access Protocol*) que resulta muy complejo (y computacionalmente pesado) porque está definido sobre la pila completa de niveles OSI. Como alternativa a DAP para acceder a directorios de tipo X.500, LDAP (*Lightweight Directory Access Protocol*) ofrece un protocolo "ligero" casi equivalente, pero mucho más sencillo y eficiente, diseñado para operar directamente sobre TCP/IP. Actualmente, la mayoría de servidores de directorio X.500 incorporan LDAP como uno de sus protocolo de acceso.

LDAP permite el acceso a la información del directorio mediante un esquema cliente-servidor, donde uno o varios servidores mantienen la misma información de directorio (actualizada mediante réplicas) y los clientes realizan consultas a cualquiera de ellos. Ante una consulta concreta de un cliente, el servidor contesta con la información solicitada y/o con un "puntero" donde conseguir dicha información o datos adicionales (normalmente, el "puntero" es otro servidor de directorio).

Internamente, el modelo de datos de LDAP (derivado de X.500, pero algo restringido) define una estructura jerárquica de objetos o *entradas* en forma de árbol, donde cada objeto o entrada posee un conjunto de atributos. Cada atributo viene identificado mediante un *nombre* o acrónimo significativo, pertenece a un cierto *tipo* y puede tener uno o varios *valores* asociados. Toda entrada viene identificada unívocamente en la base de datos del directorio mediante un atributo especial denominado *nombre distinguido* o `dn` (*distinguished name*). El resto de atributos de la entrada depende de qué objeto esté describiendo dicha entrada. Por ejemplo, las entradas que describen personas suelen tener, entre otros, atributos como `cn` (*common name*) para describir su nombre común, `sn` (*surname*) para su apellido, `mail` para su dirección de correo electrónico, etc. La definición de los posibles tipos de objetos, así como de sus atributos (incluyendo su nombre, tipo, valor(es) admitido(s) y restricciones), que pueden ser utilizados por el directorio de un servidor de LDAP la realiza el propio servidor mediante el denominado *esquema* del directorio. Es decir, el esquema contiene las definiciones de los objetos que pueden darse de alta en el directorio.

El nombre distinguido de cada entrada del directorio es una cadena de caracteres formada por pares `<tipo_atributo>=<valor>` separados por comas, que representa la ruta invertida que lleva desde la posición lógica de la entrada en el árbol hasta la raíz del mismo. Puesto que se supone que un directorio almacena información sobre los objetos que existen en una cierta organización, cada directorio posee como raíz (o *base*, en terminología LDAP) la ubicación de dicha organización, de forma que la base se convierte de forma natural en el *sufijo* de los nombres distinguidos de todas las entradas que mantiene el directorio. Existen dos formas de nombrar, o estructurar, la raíz de un directorio LDAP:

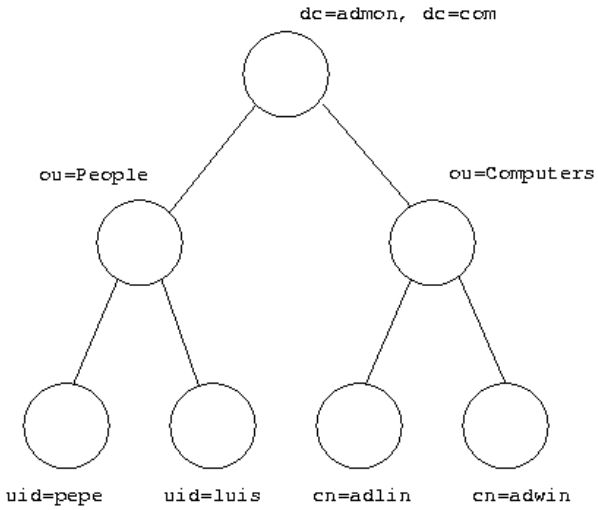
a. Nombrado "tradicional": formado por el país y estado donde se ubica la organización, seguida por el nombre de dicha organización. Por ejemplo, la raíz o base de la Universidad Politécnica de Valencia podría ser algo así: `"o=UPV, st=Valencia, c=ES"`.

b. Nombrado basado en nombres de dominio de Internet (es decir, en DNS): este nombrado utiliza los dominios DNS para nombrar la raíz de la organización. En este caso, la base de la UPV sería: "dc=upv, dc=es". Este es el nombrado que vamos a utilizar puesto que permite localizar a los servidores de LDAP utilizando búsquedas DNS.

A partir de esa base, el árbol se subdivide en los nodos y subnodos que se estime oportuno para estructurar de forma adecuada los objetos de la organización, objetos que se ubican finalmente como las

hojas del árbol. De esta forma, el nombre distinguido de cada entrada describe su posición en el árbol de la organización (y vice-versa), de forma análoga a un sistema de archivos típico, en el que el nombre absoluto (unívoco) de cada archivo equivale a su posición en la jerarquía de directorios del sistema, y vice-versa. En la Figura~ Figura 2.1. Estructura de directorio del dominio admon.com. se muestra un ejemplo de un directorio sencillo (dos usuarios y dos equipos) de la organización `admon.com`.

Figura 2.1. Estructura de directorio del dominio `admon.com`.



De acuerdo con dicha figura, la entrada correspondiente al usuario "pepe" tendría como nombre distinguido "`uid=pepe, ou=People, dc=admon, dc=com`". Al margen de ese identificador único, cada entrada u objeto en el directorio puede tener, como hemos dicho, un conjunto de atributos tan descriptivo como se desee. Cada objeto necesita, al margen de su nombre distinguido, su "clase de objeto", que se especifica mediante el atributo `ObjectClass` (un mismo objeto puede pertenecer a diferentes clases simultáneamente, por lo que pueden existir múltiples atributos de este tipo para un mismo objeto en el directorio). Este atributo especifica implícitamente el resto de atributos de dicho objeto, de acuerdo con la definición establecida en el esquema. Siguiendo con el ejemplo anterior, a continuación se muestra un subconjunto de los atributos del usuario "pepe":

```
dn: uid=pepe, ou=People, dc=admon, dc=com
objectClass: person
cn: Jose García
sn: García
description: alumno
mail: pepe@admon.com
```

El formato en el que se han mostrado los atributos del objeto se denomina LDIF (*LDAP Data Interchange Format*), y resulta útil conocerlo porque es el formato que los servidores LDAP (y OpenLDAP entre ellos) utilizan por defecto para insertar y extraer información del directorio.

Puesto que nosotros vamos a utilizar el directorio con un uso muy específico (centralizar la información administrativa de nuestro dominio para autentificar usuarios de forma global) deberíamos asegurarnos que en el esquema de nuestro directorio existen los tipos de objetos (y atributos) adecuados para ello. Afortunadamente, OpenLDAP posee por defecto un esquema suficientemente rico para cubrir este cometido. Por ejemplo, cada usuario puede definirse mediante un objeto de tipo `posixAccount`, que posee entre otros atributos para almacenar su UID, GID, contraseña, etc. A título de curiosidad, la entrada en el esquema que define el atributo para la contraseña (`userPassword`) se muestra a continuación:

```
attributetype (2.5.5.35 NAME 'userPassword'
     EQUALITY octetStringMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.40{128})
```

2.4. Configuración Básica de OpenLDAP

La utilización de LDAP para centralizar las labores administrativas (es decir, la creación de un "dominio Linux") tiene dos partes bien diferenciadas: primero hay que instalar y configurar uno o varios ordenadores del futuro dominio como *servidores de LDAP* y el resto de ordenadores del dominio como *clientes* de dicha información. Una vez conseguido este objetivo, el segundo paso es configurar los clientes para que utilicen a los servidores como fuente de información administrativa (cuentas de usuarios y grupos, contraseñas, hosts, etc.) en vez de (o mejor dicho, además de) sus ficheros de configuración locales.

En ambos casos, es importante resaltar que parte de la configuración que se describe a continuación es *dependiente* del software de LDAP concreto que se va a utilizar (en este caso, OpenLDAP) y de la versión de UNIX utilizada (RedHat Linux). La configuración de otros servidores de LDAP, en la misma u otras versiones de UNIX, puede ser distinta a la que aquí se va a exponer.

2.4.1. Configuración de OpenLDAP con Servidor Unico

En el caso más sencillo, nuestra red puede requerir un solo servidor de LDAP, siendo el resto de los ordenadores clientes del mismo. Si este es el caso, esta sección expone los pasos a seguir para configurar el servidor y los clientes.

2.4.1.1. Primer Paso: Instalación del Servidor

El paquete que incorpora el servidor de OpenLDAP se denomina `openldap-servers` y puede encontrarse en la distribución correspondiente de RedHat/Fedora Linux.

Una vez descargado e instalado, este paquete instala los ficheros de configuración por defecto bajo el directorio `/etc/openldap`, crea un directorio vacío denominado `/lib/var/ldap` para albergar la base de datos con la información del directorio y sus índices, y finalmente incorpora el servicio o "demonio" de LDAP, denominado **slapd**. Curiosamente, el nombre del servicio presente en `/etc/rc.d/init.d` , y que utilizaremos para iniciar y parar el demonio, se denomina `ldap` y no `slapd`.

2.4.1.2. Segundo Paso: Configuración del Servidor

La configuración del servicio `slapd` se realiza en `/etc/openldap/slapd.conf` fundamentalmente. Este fichero contiene referencias a los demás ficheros necesarios para el servicio `slapd` (como por ejemplo, las definiciones del esquema), y un conjunto de secciones donde se describen los directorios de los que se mantiene información. Es incluso posible almacenar los directorios en bases de datos de distintos formatos internos y definir opciones específicas diferentes para cada una. En el caso más sencillo, sin embargo, tendremos un único directorio almacenado en una base de datos en el formato por defecto (lbdm), cuyas opciones no modificaremos.

De hecho, de los numerosos parámetros que pueden configurarse en este fichero, sólo vamos a comentar los estrictamente necesarios para configurar un servidor básico de LDAP que luego haga de servidor de un dominio Linux. Para configurar cualquier otro parámetro avanzado, se recomienda la lectura previa del documento *"OpenLDAP Administrator's Guide"* [http://www.openldap.org/doc/admin22/].

En definitiva, los cinco parámetros fundamentales que es necesario configurar son los siguientes (el resto pueden dejarse con sus opciones por defecto):

1. **Sufijo**. Este es el sufijo de las entradas del directorio, es decir, lo que hemos denominado base o raíz del directorio. Por ejemplo, para el dominio "admon.com" deberíamos configurar:

   ```
   suffix  "dc=admon, dc=com"
   ```

2. **Directorio de la(s) base(s) de datos**. Se especifica mediante la directiva siguiente:

   ```
   directory /var/lib/ldap
   ```

3. **Cuenta del administrador** (del directorio). Esta es la cuenta del usuario administrador del directorio, lógicamente en formato de LDAP. Las credenciales que se sitúan en esta opción (y la siguiente) tienen validez independientemente de que este usuario exista realmente en la base de datos del directorio o no. El nombre por defecto es "manager", pero si queremos lo podemos cambiar por "root" (a la UNIX):

   ```
   rootdn  "cn=root, dc=admon, dc=com"
   ```

   Sin embargo, no hay que confundir a este usuario con el usuario "root" del sistema Linux donde estamos instalando el servidor de LDAP. Más adelante, si queremos, podemos hacerlos coincidir.

4. **Contraseña del administrador**. Cuando las operaciones a realizar en la base de datos no permitan una conexión anónima (sin acreditarse), y en concreto, cuando necesiten del usuario "administrador del directorio" definido arriba, dichas operaciones deben acompañarse de la contraseña de esta cuenta, que se especifica en la siguiente opción:

   ```
   rootpw  <CONTRASEÑA>
   ```

   Hay que tener en cuenta que la contraseña que pongamos aquí será visible por cualquier usuario que pueda leer el fichero (aunque por defecto, éste es un permiso sólo de root). Por tanto, es conveniente ponerla cifrada. Para ello, si queremos utilizar la misma contraseña que tiene "root", podemos por ejemplo copiar su contraseña cifrada del archivo `/etc/shadow` y pegarla así:

   ```
   rootpw  {crypt}<CONTRASEÑA_CIFRADA>
   ```

   O, aún mejor, podemos crear una contraseña nueva mediante la orden:

   ```
   slappasswd -h {MD5}
   ```

   Que nos pide una contrseña y nos la muestra cifrada. Luego simplemente copiamos el resultado de dicha orden tras la cadena, cambiando en este caso "{crypt}" por "{MD5}" (no debe haber espacios en blanco entre la cadena que indica el tipo de cifrado y la contraseña).

5. **Niveles de acceso**. Esta opción permite especificar, con un grano muy fino, a quién se da el permiso para leer, buscar, comparar, modificar, etc., la información almacenada en el directorio. Para ello se utilizan una o múltiples reglas de control de acceso, cuya sintaxis se muestra a continuación:

   ```
   access to <what> [by <who> <acess_control>]+
   ```

   donde:

   - `<what>` es una expresión que especifica a qué datos del directorio se aplica la regla. Existen tres opciones: (1) puede indicarse todo el directorio mediante un asterisco (`*`), (2) un subcon-

junto de entradas cuyo nombre distinguido contiene un cierto sufijo (por ejemplo, `dn=".*, dc=admon, dc=com"`) o (3) un atributo concreto de dichos objetos (por ejemplo, `dn=".*, ou=People, dc=admon, dc=com" attr=userPassword`).

- `<who>` indica a quién (a qué *usuario(s)*) se especifica la regla. Puede tomar diferentes valores, siendo los más comunes los siguientes: `self` (el propietario de la entrada), `dn="..."` (el usuario representado por el nombre distinguido), `users` (cualquier usuario acreditado), `anomymous` (cualquier usuarios no acreditado) y `*` (cualquier usuario).

- `<access_control>` indica qué operación concede la regla: `none` (sin acceso), `auth` (utilizar la entrada para validarse), `compare` (comparar), `search` (búsqueda), `read` (lectura), y `write` (modificación).

Para entender correctamente la semántica de las reglas de control de acceso, es imprescindible conocer el método que sigue `slapd` para evaluarlas, cada vez que recibe una petición por parte de un cliente: primero se busca, secuencialmente, la primera regla cuya expresión `<what>` incluye la entrada, o entradas, afectadas por la petición. Dentro de ella, se busca secuencialmente la primera expresión `<who>` que incluye al *usuario* que ha realizado la petición desde el cliente. Una vez encontrado, si el nivel de acceso expresado por `<access_control>` es mayor o igual al requerido por la petición, entonces ésta se concede, y en caso contrario se deniega. Si no existe ninguna expresión `<who>` que incluya al usuario, o bien no existe ninguna regla cuya expresión `<what>` incluya la información afectada por la petición, se deniega la petición.

Por tanto, el orden en que aparecen las reglas en el fichero, y el orden interno de las cláusulas "by" dentro de cada regla, es relevante: si varias reglas afectan a los mismos objetos, las reglas más específicas deberían ubicarse antes en el fichero; y, dentro de una regla, si incluimos varias cláusulas `by`, también debemos situar las más específicas primero.

Un ejemplo razonable de reglas de acceso podría ser el siguiente:

```
access to dn=".*,ou=People,dc=admon,dc=com" attr=userPassword
       by self write
       by dn="cn=root,dc=admon,dc=com" write
       by * auth

access to dn=".*,ou=People,dc=admon,dc=com"
       by dn="cn=root,dc=admon,dc=com" write
       by * read

access to *
       by dn="cn=root,dc=admon,dc=com" write
       by * read
```

En este ejemplo, la primera regla de acceso permite a cada usuario a cambiarse su propia contraseña (la contraseña es el atributo `userPassword` en los objetos de tipo usuario, que se sitúan por defecto dentro de la unidad organizativa "People"), al administrador cambiar la de cualquier usuario y al resto de usuarios sólo pueden utilizar este campo para autentificarse. La segunda regla de acceso se aplica al resto de los atributos de las cuentas de usuario, y permite al administrador su cambio y al resto sólo su consulta (de esta forma evitamos que un usuario pueda cambiarse su UID, GID, etc.). Finalmente, la tercera regla permite al administrador cambiar cualquier entrada del directorio y al resto de usuarios sólo leerlas.

2.4.1.3. Tercer Paso: Comprobación de la configuración

Una vez realizada la configuración del apartado anterior, ya estamos en condiciones de iniciar el servicio `slapd` y comprobar que, de momento, todo funciona correctamente. Para ello:

```
service ldap start
```

Si el servicio ha arrancado correctamente, y a pesar de que actualmente el directorio está vacío, deberíamos ser capaces de preguntar al menos por un tipo de atributo denominado "namingContexts". Para ello, utilizamos la orden `ldapsearch`:

```
ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
```

Si todo funciona bien, el resultado debería ser algo así:

```
#
# filter: (objectclass=*)
# requesting: namingContexts
#
#
# dn: namingContexts: dc=admon, dc=com
# search result
search: 2
result: 0 success
# numResponses: 2
# numEntries: 1
```

Es imprescindible que en la salida por pantalla aparezca en sufijo del dominio (`dc=admon,dc=com` en el ejemplo). Si no es así, lo más probable es que las reglas de control de acceso especificadas en el fichero de configuración de `slapd` no admitan la lectura del directorio a usuarios no autentificados (en la orden anterior estamos accediendo sin credenciales conocidas, es decir, de forma anónima). Para acceder con credenciales concretas, se utilizan las opciones `-D` y `-W`, como por ejemplo:

```
-D "cn=root,dc=admon,dc=com" -W
```

que utiliza el `dn` especificado y nos pide su contraseña de forma interactiva.

Una vez el servicio funciona correctamente, debemos iniciarlo por defecto en los niveles de ejecución 3 y 5. Para ello puede utilizarse la herramienta gráfica **serviceconf** o, en línea de órdenes:

```
chkconfig --level 35 ldap on
```

2.4.1.4. Cuarto Paso: Configuración de los Clientes

La configuración de los clientes de OpenLDAP se realiza alternativamente en dos ficheros de configuración: `/etc/openldap/ldap.conf` y `/etc/ldap.conf`. El primero sirve de fichero de configuración por defecto para todas las herramientas clientes de OpenLDAP, mientras que el segundo incluye opciones de configuración más específicas para autentificación, gestión de contraseñas, conexiones cifradas, etc.

En principio, de momento sólo necesitaríamos configurar un par de opciones del primero de ambos ficheros en todos los ordenadores clientes, incluyendo el servidor o servidores de LDAP:

```
BASE dc=admon,dc=com
HOST adlin.admon.com
```

Es importante hacer notar que OpenLDAP aún no soporta la localización del servidor basada en registros de servicio de DNS (como es el caso del Directorio Activo de Windows 2000). Por tanto, en la opción `HOST` es necesario especificar o bien una dirección IP o bien un nombre de ordenador que pueda resolverse correctamente sin utilizar el propio directorio (por ejemplo, que esté dado de alta en el fichero `/etc/hosts` del ordenador cliente, o que pueda resolverse correctamente mediante una con-

sulta DNS).

2.4.1.5. Quinto Paso: Migración de la Información del Servidor

El siguiente y último paso consiste en añadir al directorio, que aún está vacío, toda la información presente en el ordenador que está haciendo de servidor. Esto incluye todos los recursos dados de alta en sus ficheros de configuración, tales como cuentas de usuarios, grupos, ordenadores, etc., así como diferentes "contenedores" o "unidades organizativas" (*OrganizationalUnits*) que distribuyen los objetos de forma racional.

Para ello, OpenLDAP incorpora un conjunto de herramientas que pueden utilizarse para realizar esta migración de forma automática. El paso previo para ello es editar el fichero `/usr/share/openldap/migration/migrate_common.ph`. En concreto, tenemos que editar las dos directivas siguientes:

```
$DEFAULT_MAIL_DOMAIN = "admon.com";
$DEFAULT_BASE = "dc=admon,dc=com";
```

Una vez modificadas ambas, tenemos diferentes opciones para incorporar la información del sistema al directorio. Entre ellas, la más recomendable es realizar la migración por partes, añadiendo primero la "base" (es decir, las entradas correspondientes a la organización y sus unidades organizativas por defecto) y posteriormente migrando los usuarios, grupos, hosts, etc., que se ubicarán dentro de dichas unidades.

Para todo ello existen *scripts* de Perl en el directorio mencionado arriba (donde se ubica `migrate_common.ph`), que podemos utilizar de la siguiente forma (en todo el proceso, el servicio `ldap` debe estar ejecutándose):

- Para la migración de la base, se exporta primero sus objetos a un fichero, en formato LDIF, y luego se insertan en el directorio mediante la orden **ldapadd**:

  ```
  bash# ./migrate_base.pl > /root/base.ldif
  bash# ldapadd -x -c -D "cn=root,dc=admon,dc=com"
  -W -f /root/base.ldif
  ```

  Nótese que con la opción `-D` estamos acreditándonos, para la operación, como el usuario "administrador del directorio", y con la opción `-W` le decimos a la orden que nos pida la contraseña de dicho usuario de forma interactiva. La opción `-c` consigue que **ldapadd** siga insertando registros a pesar de que se produzcan errores en alguna inserción (cosa que suele ocurrir con el primer registro).

- Para la migración de los usuarios:

  ```
  bash# ./migrate_passwd.pl  /etc/passwd > /root/usuarios.ldif
  ```

  Una vez copiados todos los usuarios en format LDIF, tenemos que editar el fichero `usuarios.ldif` y eliminar todos los registros que hacen referencia a usuarios especiales de Linux, incluyendo a "root" (no se recomienda en general exportar la cuenta de "root" mediante LDAP, por cuestiones de seguridad). En definitiva, sólo deberían quedar en el fichero los registros asociados a los usuarios comunes que hemos añadido al sistema. Una vez editado el fichero, añadimos los registros al directorio:

  ```
  bash# ldapadd -x -c -D "cn=root,dc=admon,dc=com"
  -W -f /root/usuarios.ldif
  ```

- Para la migración de los grupos:


```
bash# ./migrate_group.pl  /etc/group > /root/grupos.ldif
```

Como con los usuarios, eliminamos del fichero `grupos.ldif` los grupos especiales, y dejamos sólo los grupos privados de los usuarios comunes que hemos añadido al sistema. Tras la modificación, añadimos esos grupos al directorio:

```
bash# ldapadd -x -c -D "cn=root,dc=admon,dc=com"
-W -f /root/grupos.ldif
```

- Y así sucesivament con hosts, servicios, etc. De todas formas, para nuestros propósitos de uso del directorio, con la migración hecha hasta aquí resultaría suficiente.

A partir de este momento, pueden utilizarse las utiliadades **ldapadd** para añadir entradas, **ldapmodify** y **ldapmodrdn** para modificar entradas o nombres relativos de entrada (el nombre distinguido relativo de una entrada es el primer campo del nombre distinguido de dicha entrada), **ldapdelete** para eliminar entradas, **ldappasswd** para modificar la contraseña de una entrada y la ya mencionada **ldapsearch** para buscar entradas, desde cualquiera de los clientes. Se recomienda visitar las páginas de manual correspondientes para más información.

Es importante recordar que las órdenes de añadir y modificar entradas esperan recibir la información en formato LDIF. Por ejemplo, para añadir una entrada de grupo denominada "alumnos" con GID 1000 deberíamos crear primeramente un archivo (`alumnos.ldif`) con el siguiente contenido (y al menos una línea en blanco al final):

```
dn: cn=alumnos,ou=Group,dc=admon,dc=com
objectclass: posixGroup
objectclass: top
cn: alumnos
userPassword: {crypt}x
gidNumber: 1000
```

Y posteriormente añadirlo al directorio mediante la orden:

```
bash# ldapadd -x -D "cn=root,dc=admon,dc=com"
-W -f alumnos.ldif
```

Evidentemente, esto resulta muy tedioso y es fácil equivocarse. Por este motivo, se recomienda la utilización de herramientas gráficas que faciliten la labor de crear, modificar y borrar entradas en el directorio. En Sección 2.6 se amplía este aspecto.

2.4.2. Configuración de OpenLDAP con Múltiples Servidores

Si las necesidades de nuestra red y/o de nuestro directorio hacen aconsejable mantener más de un servidor LDAP (por ejemplo, para poder equilibrar la carga de las consultas, y por aspectos de tolerancia a fallos) podemos configurar varios servidores LDAP que mantengan imágenes sincronizadas de la información del directorio.

Para mantener el directorio replicado en diferentes servidores, OpenLDAP propone un esquema de maestro único y múltiples esclavos. Este esquema funciona de la siguiente forma: cada vez que se produce un cambio en el directorio del servidor maestro, el servicio `slapd` escribe dicho cambio, en formato LDIF, en un fichero local de registro (es decir, *log file* o cuaderno de bitácora). El servidor maestro inicia otro servicio denominado `slurpd` que, cada cierto tiempo se activa, lee dichos cambios e invoca las operaciones de modificación correspondientes en todos los esclavos. En cambio, si un esclavo recibe una operación de modificación directa por parte de un cliente, ésta debe redirigirse auto-

máticamente al servidor maestro.

Evidentemente, este esquema sólo funciona si todos los servidores (maestro y esclavos) parten de un estado del directorio común. Por ese motivo, es necesario copiar manualmente la base de datos del directorio (es decir, el contenido del directorio `/var/lib/openldap` del servidor maestro) a cada esclavo, estando los servicios `slapd` parados en todos ellos, por supuesto.

La configuración del servidor maestro y de cada esclavo sería la siguiente:

A. **Servidor maestro**. En el archivo de configuración `/etc/openldap/slapd.conf` hay que añadir las siguientes líneas por cada servidor esclavo:

```
replica  host=esclavo.admon.com:389
         binddn="cn=Replicator,dc=admon,dc=com"
         bindmethod=simple
         credentials=CONTRASEÑA_PLANA
```

   Y además hay que decirle a `slapd` en qué fichero de registro tiene que escribir los cambios:

```
replogfile   /var/lib/openldap/master-slapd.replog
```

B. **Servidor esclavo**. Por una parte, en el servidor esclavo hay que configurar el archivo `/etc/openldap/slapd.conf` de la misma forma que en el maestro (ver Sección 2.4.1.2), exceptuando las líneas expuestas en el apartado anterior, que sólo corresponden al maestro.

   Por otra parte, hay que incluir en dicho fichero las siguientes opciones:

```
rootdn     "cn=Replicator,dc=admon,dc=com"
updatedn   "cn=Replicator,dc=admon,dc=com"
updateref  ldap://maestro.admon.com
```

   La opción `updatedn` indica la cuenta con la que el servicio `slurpd` del servidor maestro va a realizar las modificaciones en la réplica del esclavo. Como puede comprobarse, hemos establecido que esta cuenta sea también el "rootdn" del servidor esclavo. Esa es la forma más sencilla de asegurar que este usuario tendrá permisos para modificar el directorio en este servidor (si no fuera así, deberíamos asegurarnos que esta cuenta tuviera concedido permiso de escritura en el directorio del servidor esclavo, en directiva "access" correspondiente). Por su parte, `updateref` indica al servidor esclavo que cualquier petición directa de modificación que venga de un cliente debe ser redireccionada al servidor maestro del directorio.

2.5. Implementación de un Dominio Linux con OpenLDAP

Una vez configurado el servidor de LDAP para almacenar la información del directorio, y los clientes de LDAP para poder preguntar por ella, el siguiente y último paso para organizar un dominio Linux con LDAP es conseguir que el directorio sea utilizado por todos los ordenadores (servidores y clientes) como fuente de información administrativa, añadida a los propios ficheros de configuración locales presentes en cada ordenador.

En principio, la información administrativa que tiene sentido centralizar en un dominio Linux se reduce prácticamente a cuentas de usuario (incluyendo contraseñas) y cuentas de grupo\footnote{Las cuentas de ordenadores del directorio, es decir, la centralización del fichero `/etc/hosts` del servidor LDAP, pueden obviarse si ya disponemos de resolución de nombres basada en DNS.}. En conjunto, la información almacenada en ambos tipos de cuentas permite autentificar a un usuario cuando éste desea iniciar una sesión interactiva en un sistema Linux y, en el caso de que la autentificación sea positi-

va, crear el contexto de trabajo inicial (es decir, el proceso *shell* inicial) para ese usuario. Manteniendo ambos tipos de cuentas en el directorio permitiría una gestión completamente centralizada de los usuarios del dominio.

Internamente, este proceso de autentificación y creación del contexto inicial que Linux lleva a cabo cuando un usuario desea iniciar una sesión interactiva utiliza dos bibliotecas distintas:

A. **PAM** (*Pluggable Authentication Module*) es una biblioteca de autentificación ge\-né\-ri\-ca que cualquier aplicación puede utilizar para validar usuarios, utilizando por debajo múltiples esquemas de autentificación alternativos (ficheros locales, Kerberos, LDAP, etc.). Esta biblioteca es utilizada por el proceso de "login" para averiguar si las credenciales tecleadas por el usuario (nombre y contraseña) son correctas.

B. **NSS** (*Name Service Switch*) presenta una interfaz genérica para averiguar los parámetros de una cuenta (como su UID, GID, *shell* inicial, directorio de conexión, etc.), y es utilizada por el proceso de "login" para crear el proceso de atención inicial del usuario.

La ventaja fundamental de ambas bibliotecas consiste en que pueden reconfigurarse dinámicamente mediante ficheros, sin necesidad de recompilar las aplicaciones que las utilizan. Por tanto, lo único que necesitamos es reconfigurar ambas para que utilicen el servidor LDAP además de los ficheros locales (`/etc/passwd`, etc.) de cada ordenador.

En RedHat/Fedora Linux, esta configuración es muy sencilla. Primero instalamos (si no lo está ya) un paquete denominado `nss_ldap`, que contiene los módulos de LDAP para PAM y NSS. A partir de ahí, ejecutamos la herramienta de configuración **authconfig**, que configura automáticamente los ficheros de PAM y NSS con los mecanismos de autentificación disponibles. En nuestro caso, basta con indicar, en dos ventanas sucesivas, que nuestro método de obtención de informacion y de autentificación está basado en LDAP, indicando la dirección IP del servidor y el sufijo del directorio (en formato LDAP, claro). En principio, no debemos activar la casilla de "Utilizar TLS" (que activaría conexiones seguras) ya que no hemos activado este tipo de conexiones en el servidor.

Es importante recordar que debemos realizar esta configuración en todos los clientes primero, y sólo iniciarla en el servidor cuando hayamos asegurado que todo funciona correctamente. En cualquier caso, no resulta imprescindible configurar el servidor como cliente, si siempre incluimos los usuarios y grupos nuevos tanto en el directorio como en los ficheros locales del mismo (o bien si dichos usuarios nunca van a trabajar en el servidor).

La comprobación desde cualquier cliente que el dominio funciona es muy sencilla. Simplemente ejecutamos:

```
getent passwd
getent group
```

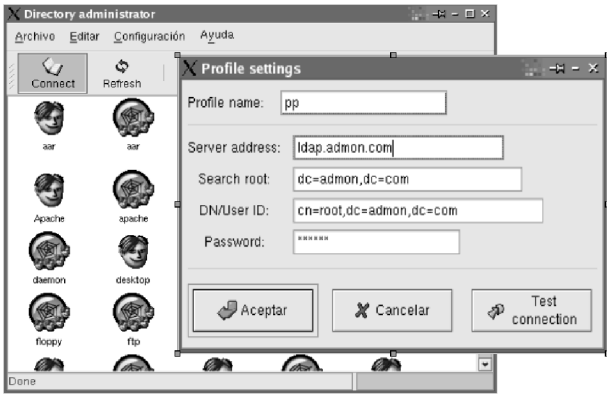
Y comprobamos que nos devuelve la lista completa de usuarios y grupos del servidor. En realidad, esta forma comprobaría únicamente el funcionamiento correcto de NSS sobre LDAP. Para una comprobación completa (PAM+NSS), la manera más efectiva es intentar iniciar una sesión en un cliente con un usuario que sólo esté definido en el servidor. Hay que recordar que el directorio de conexión del usuario debe existir en el ordenador cliente (localmente, o bien montado por NFS).

2.6. Herramientas Gráficas de Administración

Entre las diferentes herramientas gráficas con las que se puede manipular un directorio de tipo LDAP en Linux, hemos elegido una denominada *"Directory Administrator"* [http://diradmin.open-it.org] por su sencillez y comodidad.

Una vez instalado, invocamos su ejecutable (**directory_adminstrator**). La primera vez que lo ejecutamos, lanza un asistente que nos pide la información sobre el servidor de LDAP, la base del directorio, una credencial para conectarnos ("dn" y contraseña), etc. Una vez terminado el asistente, debería aparecer la ventana principal, donde vemos un objeto por cada usuario y grupo dado de alta en el directorio. Una muestra de su interfaz la tenemos en la Figura Figura 2.2. Vista de la herramienta Directory Administrator..

Figura 2.2. Vista de la herramienta Directory Administrator.



Antes de seguir, tenemos que ir a las opciones de configuración para activar la codificación de contraseñas "md5" y para desactivar el uso del atributo "`authPassword`" para almacenar la contraseña.

A partir de ese momento, ya estamos en condiciones de añadir, modificar y borrar usuarios y grupos del directorio, mediante una simple interacción con la herramienta. Si esta herramienta se convierte en la herramienta fundamental de gestión de usuarios y grupos en el dominio Linux, necesitamos tener en mente un par de precauciones: en primer lugar, si deseamos mantener la filosofía de grupos privados, debemos crear el grupo privado de un usuario *antes* que el propio usuario, ya que en este último paso sólo podemos seleccionar su grupo primario. Y en segundo lugar, tendremos que gestionar manualmente los directorios de conexión de los usuarios que creemos.

3

Sistema de Archivos en Red (NFS)

Tabla de contenidos

3.1. Introducción

Como es bien sabido por cualquier usuario de Linux, un sistema Linux puede trabajar únicamente con una jerarquía de directorios, de tal forma que si se desea acceder a distintos sistemas de archivos (particiones de discos, cd-roms, disquetes, etc.), todos ellos deben montarse primero en algún punto de dicha jerarquía.

Siguiendo la misma filosofía, *Network File System* (NFS) es un servicio de red que permite a un ordenador cliente montar y acceder a un sistema de archivos (en concreto, un *directorio*) remoto, exportado por otro ordenador servidor. Este capítulo explica las bases de cómo exportar directorios por NFS desde un sistema Linux y cómo acceder a ellos desde otros sistemas a través de la red.

3.2. Acceso a directorios remotos mediante NFS

Comenzaremos esta sección con un ejemplo. Supóngase que una máquina denominada `faemino` desea acceder al directorio `home/ftp/pub/` de la máquina `cansado`. Para ello, debería invocarse el siguiente mandato (en `faemino`):

```
mount -t nfs  cansado:/home/ftp/pub  /mnt
```

Este mandato indica que el directorio `/home/ftp/pub/`, que debe ser *exportado* por el ordenador `cansado`, va a *montarse* el directorio local `/mnt/` de `faemino`. Es importante tener en cuenta que este directorio local debe existir previamente para que el montaje pueda realizarse. La opción `-t nfs` indica a **mount** el tipo de sistema de archivos que va a montar, aunque en este caso podría omitirse, ya que mount detecta por el carácter que se trata de un montaje remoto (por NFS) gracias al carácter '`:`' en la especificación del "origen" ("`cansado:/home/ftp/pub`").

Una vez el montaje se ha realizado, cualquier acceso a archivos o directorios dentro de `/mnt` (lectura, escritura, cambio de directorio, etc.) se traduce de forma transparente a peticiones al ordenador servidor (`cansado`), que las resolverá y devolverá su respuesta al cliente, todo a través de la red. Es decir, el montaje de directorios mediante NFS permite trabajar con archivos remotos exactamente igual que si fueran locales, aunque lógicamente con una menor velocidad de respuesta.

3.3. Usos típicos de NFS

En general, NFS es muy flexible, y admite las siguientes posibilidades (o escenarios):

- Un servidor NFS puede exportar más de un directorio y atender simultáneamente a varios clientes.
- Un cliente NFS puede montar directorios remotos exportados por diferentes servidores.
- Cualquier sistema UNIX puede ser a la vez cliente y servidor NFS.

Teniendo esto en cuenta, existen varios usos típicos de NFS donde este servicio muestra su utilidad (entre otros):

A. **Directorios de conexión (home) centralizados**. Cuando en una red local de máquinas Linux se desea que los usuarios puedan trabajar indistintamente en cual\-quiera de ellas, es apropiado ubicar los directorios de conexión de todos ellos en una misma máquina y hacer que las demás monten esos directorios mediante NFS.

B. **Compartición de directorios de uso común**. Si varios usuarios (desde distintas máquinas) trabajan con los mismos archivos (de un proyecto común, por ejemplo) también resulta útil compartir (exportar+montar) los directorios donde se ubican dichos archivos.

C. **Ubicar software en un solo ordenador de la red**. Es posible instalar software en un directorio del servidor NFS y compartir dicho directorio vía NFS. Configurando los clientes NFS para que monten dicho directorio remoto en un directorio local, este software estará disponible para todos los ordenadores de la red.

3.4. Funcionamiento de NFS

En el sistema cliente, el funcionamiento de NFS está basado en la capacidad de traducir los accesos de las aplicaciones a un sistema de archivos en peticiones al servidor correspondiente a través de la red. Esta funcionalidad del cliente se encuentra normalmente programada en el núcleo de Linux, por lo que no necesita ningún tipo de configuración.

Respecto al servidor, NFS se implementa mediante dos *servicios* de red, denominados **mountd** y **nfsd**. Veamos qué acciones controlan cada uno de ellos:

a. El servicio **mountd** se encarga de atender las peticiones remotas de montaje, realizadas por la orden `mount` del cliente. Entre otras cosas, este servicio se encarga de comprobar si la petición de montaje es válida y de controlar bajo qué condiciones se va a acceder al directorio exportado (sólo lectura, lectura/escritura, etc.). Una petición se considera válida cuando el directorio solicitado ha sido explícitamente exportado y el cliente tiene permisos suficientes para montar dicho directorio. Esto se detalla más adelante en el documento.

b. Por su parte, y una vez un directorio remoto ha sido montado con éxito, el servicio **nfsd** se dedica a atender y resolver las peticiones de acceso del cliente a archivos situados en el directorio.

3.5. Instalación y configuración del cliente NFS

Como se ha expresado anteriormente, el cliente NFS no requiere ni instalación ni configuración. Los directorios remotos pueden importarse utilizando el mandato **mount** y el fichero asociado `/`

`etc/fstab`

En este sentido, recuérdese que en cada invocación al mandato **mount** (y/o en cada línea del fichero `/etc/fstab`) se pueden establecer opciones de montaje. En ellas se particulariza el comportamiento que tendrá el sistema de archivos una vez se haya montado en el directorio correspondiente. En el caso de NFS, las opciones más importantes son las que gobiernan el modo de fallo de las peticiones remotas, es decir, cómo se comporta el cliente cuando el servidor no responde:

a. `soft`. Con esta opción, cuando una petición no tiene respuesta del servidor el cliente devuelve un código de error al proceso que realizó la petición. El problema es que muy pocas aplicaciones esperan este comportamiento y ello puede provocar situaciones en las que se pierda información. Por tanto, no es aconsejable.

b. `hard`. Mediante esta opción, cuando el servidor no responde el proceso que realizó la petición en el cliente se queda suspendido indefinidamente. Esta es la opción que se recomienda normalmente, por ser la que esperan las aplicaciones, y por tanto más segura desde el punto de vista de los datos.

   Esta opción se puede combinar con la opción `intr`, que permite matar el proceso mediante el envío de una señal (de la forma tradicional en Linux).

A continuación se muestra un ejemplo, en el que se presenta la línea del archivo `/etc/fstab`(de la máquina `faemino`) relacionada con el ejemplo de la Sección 3.2:

```
#device                mountpoint  fs-type  options    dump  fsckorder
...
cansado:/home/ftp/pub  /mnt        nfs      defaults   0     0
```

3.6. Instalación y configuración del servidor NFS

El servidor necesita, además de los dos servicios **mountd** y **nfsd** (ambos se aúnan en el servicio común demominado **nfs**), uno más denominado **portmap** (del inglés **portmapper**), sobre el cual ambos basan su funcionamiento. Por tanto, la configuración de un servidor NFS necesita únicamente tener disponibles dichos servicios e iniciarlos en el nivel de ejecución 3 (o 5, o ambos) de la máquina.

Una vez activos los servicios NFS, el servidor tiene que indicar explícitamente qué directorios desea que se exporten, a qué máquinas y con qué opciones. Para ello existe un fichero de configuración denominado `/etc/exports`. A continuación se muestra uno de ejemplo, sobre el que se explican las características más relevantes:

```
# Directory            Clients and (options)
/tmp                   pc02??.dsic.upv.es(rw) *.disca.upv.es()
/home/ftp/pub          158.42.54.1(rw,root_squash)
/                      mio.dsic.upv.es(rw,no_root_squash)
/pub                   (rw,all_squash,anonuid=501,anongid=601)
/pub/nopublic          (noaccess)
```

Cada línea especifica un directorio que se va a exportar, junto con una lista de autorizaciones, es decir, qué ordenadores podrán montar dicho directorio y con qué opciones (desde el punto de vista del servidor). Cada elemento de la lista de ordenadores puede especificar un solo ordenador (mediante nombre simbólico o dirección IP) o un grupo de ordenadores (mediante el uso de caracteres comodín como `*' ó `?'). Cuando el ordendor/rango no se especifica (por ejemplo, en las últimas dos líneas), esto significa que el directorio correspondiente se exporta a todos los ordenadores del mundo (conectados a Internet). Por su parte, de entre las posibles opciones de montaje que, entre paréntesis,

pueden especificarse para cada ordenador/grupo, las más importantes se resumen a continuación:

`()`	Esta opción establece las opciones que NFS asume por defecto.
`ro`	El directorio se exporta como un sistema de archivos de sólo lectura (opción por defecto).
`rw`	El directorio se exporta como un sistema de archivos de lectura/escritura.
`root_squash`	Los accesos desde el cliente con UID=0 (root) se convierten en el servidor en accesos con UID de un usuario anónimo (opción por defecto)
`no_root_squash`	Se permite el acceso desde un UID = 0 sin conversión. Es decir, los accesos de root en el cliente se convierten en accesos de root en el servidor.
`all_squash`	Todos los accesos desde el cliente (con cualquier UID) se transforman en accesos de usuario anónimo.
`anonuid, anongid`	Cuando se activa la opción `root_squash` ó `all_squash`, los accesos anónimos utilizan normalmente el UID y GID primario del usuario denominado `nobody`, si éste existe en el servidor (opción por defecto). Si se desean utilizar otras credenciales, los parámetros `anonuid` y `anongid` establecen, respectivamente, qué `uid` y `gid` tendrá la cuenta anónima que el servidor utilizará para acceder contenido del directorio.
`noaccess`	Impide el acceso al directorio especificado. Esta opción es útil para impedir que se acceda a un subdirectorio de un directorio exportado.

Es importante destacar que cada vez que se modifica este fichero, para que se activen los cambios, el servidor NFS debe ser actualizado ejecutando el mandato **exportfs -ra**.

Una lista completa de las opciones de montaje y su significado pueden encontrarse en la página de manual `exports (5)` de Linux. La mayoría de estas opciones establecen como quién se comporta el proceso cliente cuando su petición de acceso llega al servidor. En principio, cada petición lleva asociada el UID y GID del proceso cliente, es decir, se comporta como sí mismo. No obstante, si está activa la opción `all_squash` (o bien el UID es cero y `root_squash` está activado), entonces el UID/GID se convierten en los del usuario anónimo. De todas formas, hay que tener en cuenta que los permisos sobre cada acceso del cliente se evalúan mediante los UID/GID que finalmente son válidos en el servidor (es decir, los originales o los anónimos, según el caso).

Es muy importante resaltar que no existe ningún proceso de acreditación de usuarios en NFS, por lo que el administrador debe decidir con cautela a qué ordenadores exporta un determinado directorio. Un directorio sin restricciones es exportado, en principio, a cualquier otro ordenador conectado con el servidor a través de la red (incluida Internet). Si en un ordenador cliente NFS existe un usuario con un UID igual a "X", este usuario accederá al servidor NFS con los permisos del usuario con el UID igual a "X" del servidor, aunque se trate de usuarios distintos.

4

El núcleo de Linux

Tabla de contenidos

4.1. Introducción

El núcleo de un sistema operativo es el programa que gobierna el hardware del ordenador y ofrece una interfaz de servicios al resto de programas (procesos) que se ejecutan en dicho ordenador. Normalmente, el núcleo es el primer código que se carga en memoria cuando encendemos el ordenador y permanece en dicha memoria mientras el sistema está en funcionamiento, entrando en ejecución cada vez que un proceso requiere uno de sus servicios o un dispositivo físico requiere su atención. Una de las diferencias fundamentales entre el núcleo del sistema y los procesos que se ejecutan "por encima" del mismo es que uno y otros se ejecutan en *modos de procesador* distintos (siempre que el hardware del ordenador soporte esta característica). El núcleo suele ejecutarse en alguno de los modos privilegiados del procesador, en el que tiene completo acceso a todo el hardware (incluyendo la programación de los dispositivos), mientras que los procesos se ejecutan en "modo usuario", en el que no pueden ejecutar un buen número de de instrucciones máquina del procesador. De esta forma, el núcleo se convierte en una especie de *intermediario* obligado en la mayoría de las acciones que los procesos ejecutan, lo que le permite implementar de forma adecuada la multitarea, repartiendo de forma equitativa y segura los recursos del sistema entre los diferentes procesos que se ejecutan en cada momento.

En el caso concreto de Linux, el tema del núcleo puede inducir a error, dado que se suele denominar Linux al sistema operativo completo (incluyendo todos los programas y utilidades que podemos ejecutar cuando instalamos el sistema) además de al núcleo en sí mismo. En otros sistemas operativos comerciales (como por ejemplo Windows 2000 o Solaris) esta distinción no es necesaria, puesto que ambos (el núcleo y dicho conjunto de utilidades y programas que se distribuyen junto con él) provienen de la misma empresa (en el ejemplo, Microsoft y SUN Microsystems, respectivamente) y se distribuyen conjuntamente. Sin embargo, en el caso de Linux esto no es así. El núcleo propiamente dicho es desarrollado por un grupo de personas (lideradas por el creador de Linux, Linus Torvalds) de forma *independiente* del resto de utilidades, herramientas y aplicaciones que tenemos disponibles cuando instalamos Linux. A este conjunto de utilidades (más un núcleo concreto) se lo conoce como una *distribución* de Linux, y existen numerosas empresas, organizaciones e incluso individuos que mantienen distintas distribuciones: Fedora, Debian, SuSe, Mandrake, Caldera, Gento, etc. Puesto que el núcleo se distribuye líbremente como código GPL, en realidad cualquiera de dichas organizaciones puede realizar modificaciones en el núcleo (y algunas lo hacen), pero Linus y su equipo siguen manteniendo el control sobre el núcleo reconocido.

De esta manera, las distribuciones y los núcleos tienen versiones distintas e independientes. Por ejemplo, en estos momentos la versión actual de Fedora Linux se denomina "Core 1", y el núcleo que instala por defecto es el 2.4.22. La nomenclatura de la versión del núcleo es significativa del estado del

mismo: el segundo número indica si la versión es estable (número par) o de desarrollo (número impar), mientras que el último dígito indica un número de orden (creciente) de la versión. Un número mayor indica un núcleo más reciente y (probablemente) con mayor y mejor soporte.

En la mayoría de situaciones, el núcleo pasa desapercibido al usuario común (incluso al administrador), puesto que los usuarios interactúan con los "programas" (o más correctamente con los *procesos*), que internamente interactúan con el núcleo. Sin embargo, existen situaciones en las que necesitamos *modificar* el soporte ofrecido por el núcleo, para conseguir mejores prestaciones, un sistema más seguro o mejor adaptado a un uso concreto, o simplemente para poder acceder a un nuevo tipo de dispositivo físico. Este capítulo explica los pasos necesarios para poder modificar el soporte del núcleo de Linux.

Los siguientes apartados explican los conocimientos mínimos imprescindibles sobre el núcleo de Linux (los ficheros binarios involucrados, el concepto de módulo de núcleo, donde encontrar los fuentes, etc.). Finalmente, se explica cómo obtener un núcleo nuevo, mediante la compilación de una versión en código fuente y su instalación en el sistema.

4.2. El código binario del núcleo

El código ejecutable del núcleo de Linux (el que se carga en memoria al arrancar el ordenador) reside en un fichero denominado `vmlinux` (o en su homólogo comprimido, `vmlinuz`). Normalmente, este fichero se encuentra en el directorio `/boot/` y suele tener como sufijo la versión del núcleo que generó el ejecutable (por ejemplo, `/boot/vmlinuz-2.4.22-1`). Cuando instalamos una distribución de Linux, el fichero binario del núcleo que incorpora la distribución es instalado en ese directorio y el cargador del sistema operativo (habitualmente, LILO o GRUB) lo carga en memoria y le ofrece el control al encender el ordenador. Este fichero binario es por tanto imprescindible para el funcionamiento del sistema. Por otra parte, si instalamos un binario nuevo (de una versión más moderna, por ejemplo), debemos reiniciar el ordenador para que vuelva a ser cargado en memoria. A diferencia de otros sistemas operativos, esta viene a ser la única situación en Linux en la que instalar un nuevo software obliga a reiniciar ("...para que los cambios tengan efecto").

En realidad, el código binario del núcleo puede dividirse entre diferentes ficheros: el fichero binario principal `vmlinuz` mencionado arriba (que es absolutamente necesario) y, opcionalmente, un conjunto de ficheros denominados *módulos de núcleo* (*kernel modules*). Los módulos de núcleo son ficheros objeto (compilados de forma especial) que incorporan funcionalidades concretas que pueden ser instaladas y desinstaladas del núcleo de Linux sin tener que generar un nuevo ejecutable `vmlinuz` e incluso sin tener que reiniciar el ordenador. Normalmente, la misma funcionalidad puede ser incorporada al núcleo (en fase de compilación) como parte del binario principal o como un módulo de núcleo.

Un buen ejemplo del uso de estos módulos es la instalación de manejadores (*drivers*) de dispositivos. Suponga que acabamos de instalar una nueva tarjeta de red en nuestro ordenador. Si el manejador de dicha tarjeta no está integrado en el binario principal del núcleo (`vmlinuz`), pero sí disponemos de él como un módulo de núcleo, podemos simplemente "cargarlo" en el núcleo mediante una orden de shell, sin tener que recompilar el `vmlinuz` y reiniciar el sistema. Los módulos de núcleo se encuentran disponibles en Linux desde la versión 1.2, y resultan una herramienta muy útil para multitud de situaciones, en especial para poder ofrecer un núcleo con un amplio soporte de manejadores, sistemas de archivos, protocolos de red, etc., sin tener que incluirlos necesariamente en el binario principal.

De hecho, el soporte de módulos está tan bien integrado en el sistema que también resulta transparente para los usuarios en la mayoría de ocasiones. Por ejemplo, si montamos por primera vez una partición de tipo FAT32 en un sistema en el que dicho sistema de archivos está soportado por módulos de núcleo, Linux cargará automáticamente dichos módulos y luego montará la partición en el directorio correspondiente, sin ninguna intervención al respecto del usuario (ni del administrador).

Hoy en día, distribuciones como Fedora incorporan un núcleo con un soporte razonable en el binario principal y un número muy elevado de módulos de núcleo que cubren buena parte del soporte extra

que podemos necesitar en el sistema. Sin embargo, existen ocasiones en las que necesitamos conseguir un soporte distinto. A continuación se enumeran algunas de ellas:

- Núcleo más pequeño. Algunos sistemas necesitan ejecutarse con poca memoria principal. Puesto que el binario principal del núcleo nunca se desaloja de memoria, en ocasiones necesitamos eliminar de dicho binario código que nuestro sistema no necesita y así conseguir más memoria para los procesos de usuario.

- Núcleo más seguro. Otra razón para eliminar código del núcleo que no necesitamos es incrementar su robustez y seguridad. Mantener más código (que no se usa) eleva la probabilidad de que nuestro sistema presente vulnerabilidades ante ataques que comprometan la seguridad del mismo.

- Núcleo más rápido. Un núcleo completamente "monolítico" (sólo el binario principal y sin módulos de núcleo) es más rápido que otro que tenga su soporte repartido entre ambos. En instalaciones en las que el hardware es lento o la velocidad del núcleo es crítica, es recomendable conseguir un núcleo monolítico (y optimizado). De esta forma conseguimos un núcleo altamente especializado, que probablemente sólo funcionará bien en el ordenador para el que ha sido compilado (u otros con idéntico hardware).

- Núcleo más portable. El caso contrario al anterior es conseguir un núcleo que funcione correctamente en el mayor número posible de ordenadores. Esto se consigue mediante un binario principal mínimo y el máximo número posible de módulos de núcleo.

4.3. El código fuente del núcleo

Como hemos visto en la sección anterior, existen ocasiones en las que es conveniente o necesario obtener un nuevo núcleo (binario principal más módulos).

Puesto que la mayoría de distribuciones incorporan los binarios de varios núcleos como ficheros preparados para instalar (por ejemplo, en formato rpm en el caso de Fedora), si sólo queremos obtener un núcleo más moderno para nuestra instalación de Linux lo más sencillo es obtener el último de dichos paquetes e instalarlo en el sistema.

Sin embargo, si desamos tener un núcleo para el que aún no hay versión binaria instalable, o simplemente deseamos cambiar el tipo de soporte del núcleo que tenemos actualmente instalado, necesitamos obtener el *código fuente* de dicho núcleo y recompilarlo.

Es posible que la distribución que estamos utilizando posea también como paquetes instalables los fuentes de varios núcleos (normalmente sólo de versiones estables). Si nos interesa uno de dichos núcleos, tendremos que obtener e instalar el paquete correspondiente, lo que normalmente instalará el código fuente bajo el directorio `/usr/src/`, en un directorio denominado `linux-(versión)/` (por ejemplo, `/usr/src/linux-2.4.22-1`).

Por el contrario, si el núcleo que buscamos no lo tenemos disponible como paquete instalable en nuestra distribución, tendremos que acudir a los lugares de internet que mantienen en línea los distintos núcleos disponibles de Linux, como por ejemplo *The Linux kernel archives* [http://www.kernel.org].

Si optamos por esta última opción, tenemos que considerar que el código fuente se distribuye normalmente en dos formatos distintos: como un fichero en formato tar comprimido que contiene todos los ficheros fuentes del núcleo, o bien como un "parche" que contiene sólo las *diferencias* respecto a otra versión anterior. Si no se es un usuario experto en aplicar parches a código fuente (se realiza mediante la orden **patch**) se recomienda la primera opción, porque aunque lógicamente el fichero a descargarse es mucho mayor, su instalación resulta mucho más sencilla. Una vez descargado el fichero, se descomprime en un directorio (`/usr/src/` es una opción razonable) y ya estamos en condiciones de comenzar la compilación del nuevo núcleo.

4.4. Compilación e instalación de un nuevo núcleo

Suponiendo que disponemos del código fuente de un núcleo de Linux bajo el directorio `/usr/src/linux/`, a continuación se listan las acciones que deben realizarse para construir un nuevo núcleo binario e instalarlo en el sistema.

1. Eliminar cualquier resto de ficheros de compilación en el núcleo, para recompilarlo desde cero.

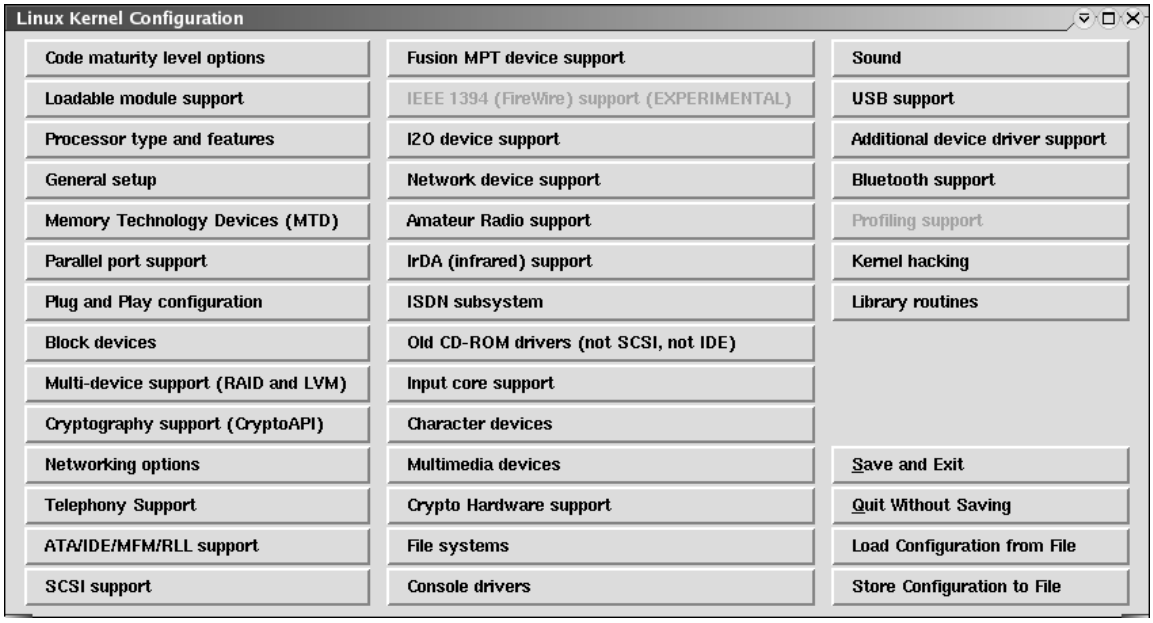
```
[root@yoda]# cd /usr/src/linux
[root@yoda]# make mrproper
```

2. Configurar el nuevo núcleo. Existen tres herramientas de configuración alternativas: una basada en texto puro (**make config**), otra en un entorno de ventanas simulado en modo texto y (**make menuconfig**) finalmente otra en modo gráfico (**make xconfig**). Si utilizamos ésta última:

```
[root@yoda]# make xconfig
```

obtenemos una ventana que ejecuta un script escrito en Tcl/Tk que visualiza un menú como el que aparece en la Figura 4.1. El menú principal de configuración del núcleo.

Figura 4.1. El menú principal de configuración del núcleo.



Como vemos, la ventana muestra multitud de botones, que ocultan a su vez menús de configuración organizados temáticamente. Al pulsar cada botón nos aparece el menú correspondiente, en donde configuraremos el soporte que deseamos para nuestro nuevo núcleo. La Figura 4.2. Uno de los submenús de configuración del núcleo. muestra, como ejemplo, el menú de configuración del tipo de procesador para el que se compilará el núcleo y sus características principales.

Figura 4.2. Uno de los submenús de configuración del núcleo.

En esta figura aparecen buena parte de las alternativas de configuración que podemos encontrarnos en todos los submenús: las opciones que aparecen en gris están habilitadas, porque son incompatibles con otras que hemos escogido (posiblemente en otros submenús). Las opciones que aparecen en negrita (opciones selecionables) pueden ser de tres tipos fundamentalmente:

- Opciones "sí/módulo/no". En este tipo de opciones podemos elegir entre tres alternativas: `y`, indicando que queremos incluir ese soporte en el binario principal del nuevo núcleo; `m`, indicando que queremos el soporte pero como módulo de núcleo; y finalmente `n`, indicando que no queremos ese soporte. Como vemos en la figura, en algunas de las opciones no existe la posibilidad de compilarla como módulo.

- Opciones múltiples. En estas opciones podemos elegir entre un conjunto de alternativas. Por ejemplo, en el tipo de procesador podemos elegir entre 16 tipos distintos. El menú con dichos tipos aparece al pulsar el botón correspondiente del submenú.

- Valor numérico. Existen opciones que permiten introducir un valor numérico concreto, que luego se pasará a los ficheros fuente involucrados como una constante de preproceso.

Cuando hemos acabado de configurar todas las opciones del núcleo que necesitamos para nuestro

nuevo núcleo, hemos de pulsar el botón de salvar los cambios (botón "`Save and Exit`" en la Figura 4.1. El menú principal de configuración del núcleo.) y salir de la herramienta de configuración.

3. Comprobar el número de versión. Antes de iniciar la compilación, es conveniente asegurarse de que el núcleo tendrá un número de versión distinto del que tenemos instalado actualmente, puesto que mientras no nos aseguremos que el nuevo núcleo funciona correctamente no es buena idea deshacerse (o sobreescribir) el antiguo. Para ello editaremos el fichero `/usr/src/linux/Makefile` y observaremos sus primeras líneas, donde se muestra la información sobre el número de versión:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 20
EXTRAVERSION = -20.9custom
```

Por ejemplo, con esta información se generaría un núcleo cuya versión sería `2.4.20-20.9custom`. En el caso de que coincidiera con un núcleo actualmente instalado y que no queremos sobreescribir, deberíamos modificar *sólo* el campo `EXTRAVERSION`. De hecho, el `Makefile` ya nos propone un sufijo denominado "`custom`" que no se encuentra en ningún núcleo "oficial".

Este número de versión (en el ejemplo, `2.4.20-20.9custom`) será el sufijo de todos los ficheros del directorio `/boot/` que dependen del núcleo concreto (`vmlinuz`, `initrd`, `System.map`, `config` y `module-info`), así como el nombre del directorio donde residirán los módulos de núcleo, debajo del directorio `/lib/modules/`.

4. Compilar el núcleo. Para compilar el binario principal y los módulos del núcleo hay que ejecutar las siguientes órdenes:

```
[root@yoda]# make dep
[root@yoda]# make bzImage
[root@yoda]# make modules
```

5. Instalar el núcleo. Los ficheros binarios del nuevo núcleo (binario principal y módulos) residen todavía en ciertos subdirectorios de `/usr/src/linux/` y no en los lugares definitivos donde el sistema podrá encontrarlos. Para llevar a cabo esta acción hay que ejecutar:

```
[root@yoda]# make modules_install
[root@yoda]# make install
```

que, siguiendo el ejemplo anterior, instalará los módulos de núcleo en el directorio denominado `/lib/modules/2.4.20-20.9custom/` y creará los siguientes ficheros y enlaces simbólicos en el directorio `/boot/`:

```
config-2.4.20-20.9
initrd-2.4.20-20.9.img
module-info-2.4.20-20.9
System.map-2.4.20-20.9
vmlinux-2.4.20-20.9
vmlinuz-2.4.20-20.9
```

```
module-info -> module-info-2.4.20-20.9
vmlinuz -> vmlinuz-2.4.20-20.9
System.map -> System.map-2.4.20-20.9
```

Además de realizar estas acciones, la instalación también realiza la configuración necesaria para que nuestro gestor de arranque (lilo o grub) sepa de la existencia del nuevo núcleo para que podamos arrancarlo en el próximo reinicio.

6. Reiniciar. Para que el núcleo nuevo sea cargado y ejecutado debemos reiniciar el sistema y asegurarnos de que en la lista de arranque de lilo o grub elegimos este núcleo y no el antiguo. En el caso de que algo salga mal (el núcleo no arranca, ciertos servicios no funcionan, etc.), deberemos reiniciar el sistema con el núcleo antiguo y probablemente volver al punto 2 para revisar la combinación de opciones con las que hemos configurado el nuevo núcleo.

7. Eliminar los ficheros objeto. Una vez el núcleo nuevo funciona correctamente y posee todas las funcionalidades que queríamos de él, podemos eliminar los objetos del directorio `/usr/src/linux/`, puesto que no los necesitamos más. Para ello ejecutaremos lo siguiente:

```
[root@yoda]# cd /usr/src/linux
[root@yoda]# make clean
```

La diferencia entre esta orden y la del punto 1 es que en este caso no se borran los ficheros de configuración del núcleo, con lo que si volvemos a ejecutar **make xconfig** tendremos en pantalla las mismas opciones con las que compilamos el núcleo la última vez.

Como alternativa, también podemos guardarnos esta configuración en un fichero, para poder recuperarla más tarde. Esto se realiza mediante los botones "`Store Configuration to File`" y "`Load Configuration from File`" de la ventana principal de configuración (ver Figura 4.1. El menú principal de configuración del núcleo.). Si elegimos esta opción (guardarnos el fichero de configuración) podemos incluso borrar el directorio completo de los fuentes, puesto que siempre podremos volver a obtenerlo más tarde si es necesario.

5

El sistema de archivos Proc

Tabla de contenidos

5.1. Introducción

En la mayoría de sistemas operativos suele ser habitual disponer de herramientas que permiten conocer el estado interno del núcleo desde nivel de usuario. Normalmente, existen diferentes herramientas para cubrir distintos aspectos de dicho estado (procesos e hilos, memoria, ficheros, interrupciones y dispositivos, etc.) y, en función del sistema operativo, la información que puede obtenerse es más o menos completa. Cualquier distribución de Linux incorpora herramientas de este tipo, tales como:

- **ps**. Visualiza los procesos en memoria el momento actual. En función de los modificadores, puede mostrar sólo los procesos lanzados en el *shell* actual o todos los procesos del sistema y, para cada uno, más o menos información. Para mostrar el máximo de información, puede utilizarse: "**ps aux**"

- **mount**. Invocado sin argumentos, muestra una lista de las particiones (locales o remotas) montadas actualmente en el sistema.

- **free**. Muestra información acerca de la memoria ocupada y libre del sistema, tanto la física como la virtual (ubicada en las particiones "*swap*").

- **uptime**. Muestra cuánto tiempo ha estado funcionando el sistema (desde el último reinicio), así como el número de usuarios conectados actualmente y la carga media del procesador en los últimos 1, 5 y 15 minutos.

A diferencia de muchos de esos sistemas operativos, Linux incorpora además una forma totalmente distinta de conocer, e incluso modificar, el estado interno del núcleo y de los procesos de usuario, mediante una interfaz de sistema de archivos denominado "`proc`". Este capítulo comenta las características fundamentales de este "sistema de archivos" y cómo podemos utilizarlo para interrogar el estado (y modificar el comportamiento) del núcleo de Linux en cualquier momento.

5.2. El sistema de archivos `/proc`

Tal como muestra el título, el sistema de archivos "proc" se monta en el directorio `/proc/` y, si listamos su contenido, veremos que se encuentra lleno de archivos y directorios. Por ejemplo:

```
[root@yoda]# ls -Cp /proc/
1/      24693/  5039/  5284/  5371/  apm          locks
10/     24696/  5048/  5289/  5373/  bus/         mdstat
11/     24697/  5113/  5303/  5385/  cmdline      meminfo
15/     24699/  5122/  5311/  5395/  cpuinfo      misc
2/      24701/  5126/  5313/  5398/  devices      modules
20554/  3/      5144/  5314/  5412/  dma          mounts@
22394/  4/      5152/  5316/  5642/  driver/      mtrr
22397/  4419/   5153/  5318/  5674/  execdomains  net/
24493/  4420/   5154/  5320/  5675/  fb           partitions
24507/  4743/   5155/  5325/  5707/  filesystems  pci
24556/  4865/   5156/  5326/  5813/  fs/          scsi/
24565/  4869/   5157/  5328/  5814/  ide/         self@
24620/  4887/   5158/  5329/  6/     interrupts   slabinfo
24653/  4906/   5170/  5331/  6912/  iomem        stat
24654/  4965/   5171/  5332/  6915/  ioports      swaps
24657/  4966/   5190/  5333/  7/     irq/         sys/
24659/  4978/   5233/  5335/  72/    kcore        sysvipc/
24685/  5/      5276/  5336/  7775/  kmsg         tty/
24688/  5015/   5279/  5348/  8/     ksyms        uptime
24689/  5029/   5282/  5367/  9/     loadavg      version
```

Sin embargo, este contenido no se encuentra guardado en ningún dispositivo físico (disco, disquete, cd, etc.), sino que es *construido* y presentado dinámicamente cada vez que le pedimos al núcleo que lo muestre, y lo mismo ocurre cuando visualizamos el contenido de sus archivos y subdirectorios. Este tipo de sistema de archivos se denomina sistema de archivos *virtual*. Si listamos de nuevo este directorio en otro momento, o en otro sistema, es posible que el listado no coincida exactamente con el del ejemplo, aunque algunos ficheros siempre serán listados; de igual forma, es posible que el *contenido* de los archivos y directorios también difiera. Ello es debido a que el contenido del directorio refleja el estado actual del núcleo de Linux, y evidentemente este estado varía con el tiempo y de un sistema a otro (por ejemplo, por disponer de hardware distinto).

Más formalmente, podemos definir `/proc/` como una *interfaz* entre el núcleo de Linux y el nivel de usuario con la forma de un sistema de archivos virtual. Así, el núcleo de Linux presenta una manera *única y homogénea* de presentar su información interna y se convierte en la alternativa a utilizar múltiples herramientas particulares como las que citábamos en el apartado anterior. La ventaja principal es que, aunque Linux incorpora muchas de ellas (en algunos casos, por compatibilidad con otras versiones de UNIX), no es imprescindible disponer de una herramienta por cada información que el núcleo pone a disposición del usuario, puesto que toda esa información está en `/proc`. Esto es especialmente útil en el caso de Linux, puesto que como veíamos en el Capítulo 4. *El núcleo de Linux*, el desarrollo del núcleo es independiente del desarrollo de las herramientas y programas que se ejecutan por encima (lo que se denomina "distribución").

Por ejemplo, veamos la información del mandato **mount**, seguida de la consulta del fichero correspondiente de `/proc/`:

```
[yoda@root]# mount
/dev/hdb1 on / type ext2 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/hdb3 on /mnt/backup type ext2 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
```

```
[yoda@root]#
[yoda@root]# cat /proc/mounts
/dev/root / ext2 rw 0 0
/proc /proc proc rw 0 0
usbdevfs /proc/bus/usb usbdevfs rw 0 0
/dev/hdb3 /mnt/backup ext2 rw 0 0
none /dev/pts devpts rw 0 0
none /dev/shm tmpfs rw 0 0
none /proc/sys/fs/binfmt_misc binfmt_misc rw 0 0
```

Como vemos, el resultado es el mismo (probablemente **mount** interroga internamente a `/proc/mounts`), aunque normalmente el mandato muestra ls información de manera algo más elaborada para el usuario. En muchos casos, la información que muestran los archivos de `/proc/` suele ser poco autoexplicativa, y es conveniente conocer de antemano el significado de los diferentes campos del "archivo" para interpretar correctamente lo que está comunicándonos el núcleo. Sin embargo, esto no es problema porque existe buena documentación al respecto, como por ejemplo la documentación de Red Hat Linux, en concreto la *"Reference Guide" de Red Hat 9* [https://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-proc.html].

Los dos siguientes apartados enseñan, respectivamente, cómo obtener y modificar la información del núcleo de Linux interactuando con el sistema de archivos `/proc/`.

5.3. Obtener información del núcleo

A pesar de que el sistema de archivos `proc` es virtual y la información que contiene se construye cada vez que el usuario realiza una operación de lectura, el nombre de los archivos y directorios que podemos encontrar en el directorio `/proc/` está predefinido por el núcleo y es bien conocido.

Esta sección presenta un repaso de los archivos y directorios más relevantes, junto con instrucciones para interpretar su contenido. Para una descripción más exhaustiva, se recomienda documentación especializada como la "*Reference Guide*" de Red Hat 9 que mencionábamos en el apartado anterior, o la propia página de manual de "proc" (**man proc**).

Algunos de los archivos más importantes son los siguientes:

1. `/proc/cmdline`. Muestra los parámetros con los que se inició el núcleo en tiempo de arranque. Por ejemplo:

   ```
   auto BOOT_IMAGE=linux ro BOOT_FILE=/boot/vmlinuz-2.4.20-20.8 hdc=ide-scsi
   root=LABEL=/
   ```

2. `/proc/cpuinfo`. Identifica el tipo de procesador del sistema y sus parámetros. Por ejemplo:

   ```
   processor       : 0
   vendor_id       : GenuineIntel
   cpu family      : 6
   model           : 7
   model name      : Pentium III (Katmai)
   stepping        : 3
   cpu MHz         : 498.858
   cache size      : 512 KB
   fdiv_bug        : no
   hlt_bug         : no
   f00f_bug        : no
   coma_bug        : no
   fpu             : yes
   ```

```
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov
pat pse36 mmx fxsr sse
bogomips        : 996.14
```

3. `/proc/filesystems`. Muestra los sistemas de ficheros que actualmente soporta el núcleo. Es necesario disponer del soporte de sistema de ficheros con el que está formateada una partición (o disquete, cd, etc.) para poder montarlo y así acceder a su contenido. Por ejemplo:

```
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   shm
nodev   pipefs
        ext2
nodev   ramfs
        iso9660
nodev   devpts
nodev   usbdevfs
nodev   usbfs
nodev   autofs
nodev   binfmt_misc
        vfat
```

   La primera columna indica si actualmente existe alguna partición montada con dicho sistema de archivos (vacío) o no (`nodev`). La segunda columna indica el tipo de sistema de archivos. Si el núcleo soporta otros sistemas de archivos como módulos que aún no se han cargado, estos no aparecen en la lista (hasta que son cargados, obviamente).

4. `/proc/meminfo`. Muestra información sobre el uso actual de la memoria RAM, indicando la cantidad de memoria utilizada en varios conceptos (compartida, en "buffers" de entrada/salida, libre, intercambio, etc.). Veamos un ejemplo:

```
        total:    used:    free:  shared: buffers:  cached:
Mem:  393965568 203403264 190562304        0  5488640 111288320
Swap: 526409728         0 526409728
MemTotal:       384732 kB
MemFree:        186096 kB
MemShared:           0 kB
Buffers:          5360 kB
Cached:         108680 kB
SwapCached:          0 kB
Active:         175636 kB
ActiveAnon:      65804 kB
ActiveCache:    109832 kB
Inact_dirty:       452 kB
Inact_laundry:       0 kB
Inact_clean:      3756 kB
Inact_target:    35968 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       384732 kB
LowFree:        186096 kB
SwapTotal:      514072 kB
SwapFree:       514072 kB
```

Las dos primeras líneas muestran un resumen del estado de la memoria RAM y de la memoria de intercambio (o *swap*), con cantidades expresadas en bytes. El resto de líneas muestran la misma información de manera más detallada y expresada en Kbytes. Los campos más relevantes son `MemTotal` (memoria física total), `MemFree` (memoria física no utilizada actualmente), `SwapTotal` (total de memoria de intercambio) y `SwapFree` (cantidad de memoria de intercambio no utilizada actualmente).

5. `/proc/modules`. Muestra un listado de los módulos de núcleo presentes en el sistema (cargados actualmente o no). Este listado puede variar sensiblemente de un sistema a otro (en un núcleo monolítico puro, estaría vacío). Por ejemplo:

```
nls_iso8859-1           3516   1 (autoclean)
nls_cp437               5148   1 (autoclean)
vfat                   13100   1 (autoclean)
fat                    38776   0 (autoclean) [vfat]
sr_mod                 18200   0 (autoclean)
ymfpci                 45516   0 (autoclean)
ac97_codec             14600   0 (autoclean) [ymfpci]
uart401                 8420   0 (autoclean) [ymfpci]
sound                  74196   0 (autoclean) [uart401]
soundcore               6500   4 (autoclean) [ymfpci sound]
binfmt_misc             7464   1
autofs                 13332   0 (autoclean) (unused)
ipt_REJECT              4024   2 (autoclean)
iptable_filter          2412   1 (autoclean)
ip_tables              15096   2 [ipt_REJECT iptable_filter]
ide-scsi               12176   0
scsi_mod              107608   2 [sr_mod ide-scsi]
ide-cd                 35808   0
cdrom                  33728   0 [sr_mod ide-cd]
CDCEther               14492   1
acm                     7904   0 (unused)
mousedev                5588   1
keybdev                 2976   0 (unused)
hid                    22340   0 (unused)
input                   5920   0 [mousedev keybdev hid]
usb-uhci               26412   0 (unused)
usbcore                78944   1 [CDCEther acm hid usb-uhci]
```

La primera columna muestra el nombre del módulo, la segunda su tamaño en memoria (en bytes), la tercera si está actualmene cargado (1) o no (0) y la cuarta y la quinta muestran información sobre si está en uso por otros módulos o no y, en su caso, cuáles son estos módulos).

6. `/proc/mounts`. Proporciona información sobre los sistemas de archivos actualmente montados en el sistema (ver ejemplo en sección anterior).

7. `/proc/swaps`. Muestra información sobre el estado de la memoria de intercambio, pero subdividido por particiones de intercambio. Conocer esta información puede ser interesante si el sistema dispone de múltiples particiones de este tipo.

8. `/proc/uptime`. Ofrece el tiempo (en segundos) desde que el sistema se inició, y la cantidad de este tipo en que el procesador ha estado ocioso. Por ejemplo:

```
3131.40 2835.67
```

9. `/proc/version`. Muestra la versión del núcleo y la del compilador de C (gcc). En el caso de Red Hat Linux, muestra también la versión de Red Hat instalada:

```
Linux version 2.4.20-20.8 (bhcompile@daffy.perf.redhat.com)
(gcc version 3.2 20020903 (Red Hat Linux 8.0 3.2-7))
#1 Mon Aug 18 14:59:07 EDT 2003
```

En cuanto a los subdirectorios más relevantes de `/proc/`, podríamos destacar los siguientes:

1. `/proc/#número#/`. Dentro de `/proc/` existe un subdirectorio por cada proceso que actualmente es conocido por el sistema. El nombre del subdirectorio es precisamente el PID (*Process IDentifier*) del proceso, es decir, el número entero positivo por el cual el núcleo de Linux identifica unívocamente a cada proceso en el sistema. Dentro del subdirectorio correspondiente a cada proceso podemos encontrar varios subdirectorios y archivos. Por ejemplo, si sabemos que el PID del proceso que ejecuta el **emacs** en el que estamos editando estas líneas es el `20540`, podemos ver su contenido:

```
[root@yoda root]# ll /proc/20540
total 0
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 cmdline
lrwxrwxrwx    1 aterrasa aterrasa        0 dic  9 09:57 cwd -> /home/aterrasa/
/DOCBOOK/comos-linux-basico
-r--------    1 aterrasa aterrasa        0 dic  9 09:57 environ
lrwxrwxrwx    1 aterrasa aterrasa        0 dic  9 09:57 exe -> /usr/bin/emacs
dr-x------    2 aterrasa aterrasa        0 dic  9 09:57 fd
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 maps
-rw-------    1 aterrasa aterrasa        0 dic  9 09:57 mem
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 mounts
lrwxrwxrwx    1 aterrasa aterrasa        0 dic  9 09:57 root -> /
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 stat
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 statm
-r--r--r--    1 aterrasa aterrasa        0 dic  9 09:57 status
```

   De estos archivos, los más importantes son:

   - `cmdline`. Contiene la línea de órdenes que inició el proceso.
   - `cwd`. Es un enlace simbólico al directorio de trabajo actual del proceso.
   - `environ`. Contiene la lista de variables de entorno del proceso.
   - `exe`. Enlace simbólico que apunta al fichero ejecutable (normalmente binario) del proceso.
   - `fd/`. Directorio que contiene enlaces simbólicos a los descriptores de fichero abiertos por el proceso actualmente.
   - `maps`. Contiene información acerca de los mapas de memoria (virtual) ocupada por el proceso, incluyendo información sobre la ubicación de sus ejecutables y bibliotecas.
   - `stat`, `statm`, y `status`. Estos archivos contienen diferentes imágenes del estado actual del proceso, incluyendo estado de la memoria, información de protección, etc.

2. `/proc/self/`. Enlace simbólico al subirectorio del proceso que actualmente está en ejecución.
3. `/proc/fs/`. Contiene información sobre los directorios que actualmente se están exportando

(sólo tiene información si el sistema es un servidor NFS).

4. `/proc/ide/`. Contiene información sobre los discos duros IDE que están conectados en el sistema, incluyendo datos de los manejadores, opciones de configuración y estadísticas de uso.

5. `/proc/net/`. Este directorio contiene información exhaustiva sobre protocolos, parámetros y estadísticas relacionadas con el acceso a la red.

6. `/proc/scsi/`. Equivale al directorio `/proc/ide/`, pero para dispositivos SCSI.

7. `/proc/sys/`. Este subdirectorio es especial y distinto del resto de directorios que hemos visto hasta ahora, puesto que no sólo permite leer informacion actual del núcleo, sino que permite *modificarla* en algunos casos. Esto se explica con más detalle en la siguiente sección.

5.4. Modificar información del núcleo

Una de las funcionalidades avanzadas de `/proc/` es, como decíamos, la posibilidad de modificar algunos de los parámetros internos del núcleo sin tener que recompilarlo y reiniciar el sistema. Esta sección presenta un pequeño resumen de cómo realizarlo.

Como cualquier programa en ejecución, el núcleo de Linux posee internamente un conjunto de variables globales (o parámetros) que reflejan y/o condicionan su funcionamiento. Algunos de estas variables, como por ejemplo los procesos que se están ejecutando, la ocupación de la memoria o los sistemas de archivos montados, pueden ser consultados por el usuario mediante la lectura de los archivos correspondientes del directorio `/proc/`, tal como veíamos en la sección anterior. Pero `/proc/` permite también la *modificación* en línea de algunas de esas variables, sin necesidad de reiniciar el sistema, de forma que podemos ajustar el comportamiento del núcleo dinámicamente. Para ello se dispone del directorio `/proc/sys/`.

5.4.1. El directorio `/proc/sys`

El directorio `/proc/sys/` contiene un subárbol de directorios y archivos en el que se organizan muchos parámetros del núcleo, subdivididos por categorías (es decir, por subdirectorios). Los archivos pueden tener permisos sólo de lectura o bien permitir su modificación, aunque sólo al administrador (`root`). Estas dos posibilidades indican, respectivamente, archivos que muestran información sobre parámetros y otros que permiten además, la modificación de estos parámetros. En este último caso, si sobreescribimos el contenido del archivo, estaremos cambiando el parámetro correspondiente del núcleo. Por ejemplo:

```
[root@yoda root]# cat /proc/sys/kernel/hostname
yoda.dsic.upv.es
[root@yoda root]# echo obiwan.dsic.upv.es > /proc/sys/kernel/hostname
[root@yoda root]# cat /proc/sys/kernel/hostname
obiwan.dsic.upv.es
```

Los subdirectorios más significativos de `/proc/sys/` son los siguientes:

- `/proc/sys/dev/`. Este directorio proporciona los parámetros de configuración de algunos dispositivos físicos conectados al sistema, como por ejemplo unidades de cdrom. Por ejemplo:

  ```
  [root@yoda root]# ls -l /proc/sys/dev/cdrom/
  total 0
  -rw-r--r--    1 root     root            0 dic  9 11:51 autoclose
  ```

```
-rw-r--r--    1 root     root            0 dic  9 11:51 autoeject
-rw-r--r--    1 root     root            0 dic  9 11:51 check_media
-rw-r--r--    1 root     root            0 dic  9 11:51 debug
-r--r--r--    1 root     root            0 dic  9 11:51 info
-rw-r--r--    1 root     root            0 dic  9 11:51 lock
[root@yoda root]# cat /proc/sys/dev/cdrom/info
CD-ROM information, Id: cdrom.c 3.12 2000/10/18

drive name:             sr0
drive speed:            40
drive # of slots:       1
Can close tray:         1
Can open tray:          1
Can lock tray:          1
Can change speed:       1
Can select disk:        0
Can read multisession:  1
Can read MCN:           1
Reports media changed:  1
Can play audio:         1
Can write CD-R:         1
Can write CD-RW:        1
Can read DVD:           0
Can write DVD-R:        0
Can write DVD-RAM:      0
```

Mediante la modificación de archivos como `autoclose` o `autoeject` (que pueden contener `0` o `1`) podemos controlar la activación o desactivación de esas características de la unidad de cd.

- `/proc/sys/fs/`. Este directorio contiene numerosos parámetros concernientes a los sistemas de archivos montados en el sistema, incluyendo cuotas, manejadores de archivo, inodos, etc:

```
[root@yoda root]# ls -l /proc/sys/fs/
total 0
dr-xr-xr-x    2 root     root            0 dic  9 12:10 binfmt_misc
-r--r--r--    1 root     root            0 dic  9 12:10 dentry-state
-rw-r--r--    1 root     root            0 dic  9 12:10 dir-notify-enable
-rw-r--r--    1 root     root            0 dic  9 12:10 file-max
-r--r--r--    1 root     root            0 dic  9 12:10 file-nr
-r--r--r--    1 root     root            0 dic  9 12:10 inode-nr
-r--r--r--    1 root     root            0 dic  9 12:10 inode-state
-rw-r--r--    1 root     root            0 dic  9 12:10 lease-break-time
-rw-r--r--    1 root     root            0 dic  9 12:10 leases-enable
-rw-r--r--    1 root     root            0 dic  9 12:10 overflowgid
-rw-r--r--    1 root     root            0 dic  9 12:10 overflowuid
dr-xr-xr-x    2 root     root            0 dic  9 12:10 quota
```

Por ejemplo, si quisiéramos incrementar la cantidad de manejadores de fichero (es decir, el número máximo de ficheros que pueden ser abiertos simultáneamente), deberíamos incrementar la cantidad que aparece en `file-max`.

- `/proc/sys/kernel/`. El contenido de este directorio incluye aspectos de configuración y parámetros que afectan directamente el funcionamiento del núcleo, como por ejemplo el comportamiento de [Ctl]-[Alt]-[Supr] para reiniciar el sistema, el nombre del ordenador (ver ejemplo arriba), el número máximo de hilos de ejecución que el núcleo puede ejecutar, etc.

- `/proc/sys/net/`. Este directorio permite ver y controlar el funcionamiento de muchos aspectos del núcleo relacionados con la red, incluyendo los diferentes protocolos queimplementa Linux (`ethernet`, `ipx`, `ipv4`, `ipv6`, etc. Por ejemplo, para permitir la retransmisión de paquetes entre

dos tarjetas de red conectadas al ordenador, haríamos lo siguiente:

```
[root@yoda root]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

- `/proc/sys/vm/`. Este subdirectorio permite la configuración del subsistema de memoria virtual del núcleo, como por ejemplo el funcionamiento del servicio (o "demonio") de intercambio (**kswapd**), la cantidad de memoria que se dedicará a buffers del núcleo, el número máximo de áreas de memoria que pueden tener los procesos, etc.

5.4.2. El mandato sysctl

Como alternativa a escribir directamente el valor (o los valores) en los archivos de `/proc/sys/`, Linux dispone de un mandato denominado **sysctl**, que puede ser utilizado para leer y modificar todos los parámetros del kernel que se ubican en ese directorio. Por ejemplo, para modificar el nombre de la máquina, como veíamos arriba, podríamos ejecutar lo siguiente:

```
[root@yoda root]# cat /proc/sys/kernel/hostname
yoda.dsic.upv.es
[root@yoda root]# sysctl -w kernel.hostname=obiwan.dsic.upv.es
kernel.hostname = obiwan.dsic.upv.es
[root@yoda root]# cat /proc/sys/kernel/hostname
obiwan.dsic.upv.es
```

La orden **sysctl** se utiliza para leer y modificar *claves* que, como vemos, coinciden con las rutas de los archivos correspondientes dentro del directorio `/proc/sys/`. Para un listado completo de las claves que podemos modificar, podemos teclear **sysctl -a**.

6

El Servicio DHCP en GNU/Linux

Tabla de contenidos

6.1. Introducción

DHCP (*Dynamic Host Configuration Protocol*) o Protocolo Dinámico de Configuración de Equipos no es un protocolo específico de Linux, sino que se trata de un estándar para cualquier tipo de sistema conectado a una red TCP/IP.

La función básica de este protocolo es evitar que el administrador tenga que configurar manualmente las características propias del protocolo TCP/IP en cada equipo. Para ello, existe en la red un sistema especial, denominado servidor DHCP, que es capaz de asignar la configuración TCP/IP al resto de máquinas presentes en la red, o clientes DHCP, cuando estos arrancan.

Entre los datos que más habitualmente proporciona el servidor a los clientes se incluyen:

- Una dirección IP por cada tarjeta de red o NIC (*Network Interface Card*) que posea el cliente.
- La máscara de subred.
- La puerta de enlace o gateway.
- Otros parámetros adicionales, como el sufijo del dominio DNS, o el propio servidor DNS.

En una red pueden convivir equipos que sean clientes DHCP con otros cuya configuración se haya establecido manualmente. Aquellos que estén configurados como clientes DHCP necesitarán encontrar en la red local un servidor DHCP para que les proporciones los parámetros TCP/IP.

Cuando un cliente arranca por primera vez, lanza por la red un mensaje de difusión, solicitando una dirección IP. Si en la red existe un solo servidor DHCP, cuando este reciba el mensaje contestará al cliente asociándole una dirección IP junto con el resto de parámetros de configuración. En concreto, el servidor DHCP puede estar configurado para asignar al cliente una dirección IP cualquiera de las que tenga disponibles, o bien para asignarle una dirección en concreto (o dirección reservada), en función de la dirección física de la tarjeta ethernet del cliente. En ambos casos, una vez el cliente recibe el mensaje del servidor, ya tiene una configuración IP con la que poder acceder a la red de forma normal.

Si en la red hay más de un servidor DHCP, es posible que dos o más servidores escuchen la petición y la contesten. Entonces, el primer mensaje que recibe el cliente es aceptado y el resto son rechazados. Es muy importante resaltar que cuando hay varios servidores DHCP en una misma red local, estos no se comunican entre ellos para saber qué direcciones IP debe asignar cada uno. Es responsabilidad de los administradores que sus configuraciones sean independientes y consistentes.

Por tanto, cuando en una misma red TCP/IP existe más de un servidor DHC, es imprescindible que estén configurados de manera que no puedan asignar la misma dirección IP a dos ordenadores distintos. Para ello basta que los rangos de direcciones IP que puedan proporcionar no tengan direcciones comunes, o, si las tienen, que estas sean direcciones reservadas.

En cualquiera de los casos anteriores, desde el punto de vista del cliente los parámetros que ha recibido se consideran una concesión, es decir, son válidos durante un cierto tiempo. Cada vez que el cliente arranca, o bien cuando se alcanza el límite de la concesión (lease time) el cliente tiene que solicitar su renovación.

El protocolo DHCP es especialmente útil cuando el parque de equipos de una organización se distribuye en varias subredes físicas, y además los equipos cambian de ubicación (de subred) con cierta frecuencia. En este caso, cambiar el equipo de sitio no supone nunca reconfigurar manualmente sus parámetros de red, sino simplemente conectarlo a la nueva red y arrancarlo.

6.2. Concesión y Renovación

Un cliente DHCP obtiene una concesión para una dirección IP de un servidor DHCP. Antes que se acabe el tiempo de la concesión, el servidor DHCP debe renovar la concesión al cliente o bien este deberá obtener una nueva concesión. Las concesiones se guardan en la base de datos del servidor DHCP aproximadamente un día después de que se agote su tiempo. Este periodo de gracia protege la concesión del cliente en caso de que este y el servidor se encuentren en diferentes zonas horarias, de que sus relojes internos no estén sincronizados o en caso de que el cliente esté fuera de la red cuando caduca el tiempo de la concesión.

La primera vez que se inicia un cliente DHCP e intenta unirse a una red, se realiza automáticamente un proceso de inicialización para obtener una concesión de un servidor DHCP:

1. El cliente DHCP solicita una dirección IP difundiendo un mensaje *DHCP Discover*.
2. El servidor responde con un mensaje *DCHP Offer* proporcionando una dirección al cliente.
3. El cliente acepta la oferta respondiendo con un mensaje *DHCP Request*.
4. El servidor envía un mensaje *DHCP Ack* indicando que aprueba la concesión.
5. Cuando el cliente recibe la confirmación entonces configura sus propiedades TCP/IP usando la información de la respuesta DHCP.

Si ningún servidor DHCP responde a la solicitud del cliente (DHCP Discover), entonces el cliente autoconfigura una dirección IP para su interfaz.

En raras ocasiones un servidor DHCP puede devolver una confirmación negativa al cliente. Esto suele ocurrir si el cliente solicita una dirección no válida o duplicada. Si un cliente recibe una confirmación negativa (*DHCP Nack*), entonces deberá comenzar el proceso de concesión.

Cuando se inicia un cliente que ya tenía concedida una dirección IP previamente, este debe comprobar si dicha dirección sigue siendo válida. Para ello, difunde un mensaje *DHCP Request* en vez de un mensaje *DHCP Discover*. El mensaje *DHCP Request* contiene una petición para la dirección IP que

se le asignó previamente. Si el cliente puede usar la dirección IP solicitada, el servidor responde con un mensaje *DHCP Ack*. Si el cliente no pudiera utilizarla porque ya no es válida, porque la este usando otro cliente o porque el cliente se ha desplazado físicamente a otra subred., entonces el servidor responde con un mensaje *DHCP Nack*, obligando al cliente a reiniciar el proceso de concesión. Si el cliente no consigue localizar un servidor DHCP durante el proceso de renovación, entonces este intenta hacer un *ping* al gateway predeterminado que se lista en la concesión actual, procediendo de la siguiente forma:

- Si tiene éxito, el cliente DHCP supone que todavía se encuentra en la red en la que obtuvo la concesión actual y la seguirá usando. En segundo plano, el cliente intentará renovar la concesión actual cuando se agote el 50\% del tiempo de la concesión asignada.
- Si falló el *ping*, el cliente supone que se desplazó a otra red y autoconfigura su dirección IP, intentando cada 5 minutos localizar un servidor DHCP y obtener una concesión.

La información de TCP/IP que se concede al cliente, deberá ser renovada por este de forma predeterminada cuando se haya agotado el 50\% del tiempo de concesión. Para renovar su concesión, un cliente DHCP envía un mensaje *DHCP Request* al servidor del cual se obtuvo la concesión. El servidor renueva automáticamente la concesión respondiendo con un mensaje *DHCP Ack*. Este mensaje contiene la nueva concesión, así como cualquier parámetro de opción DHCP. Esto asegura que el cliente DHCP puede actualizar su configuración TCP/IP s el administrador de la red actualiza cualquier configuración en el servidor DHCP.

6.3. Concepto de Ambito

En el contexto de DHCP, un ámbito (\SF{scope}) se define como una agrupación administrativa de direcciones IP que posee una serie de parámetros de configuración comunes y que se utiliza para asignar direcciones IP a clientes DHCP situados en una misma red física.

Es decir, para que un servidor DHCP pueda asignar direcciones IP a sus potenciales clientes, es necesario que defina al menos un ámbito en cada red física en la que haya clientes que atender. El administrador debe establecer para dicho ámbito sus parámetros de configuración, tales como el rango de direcciones IP que puede asignar, las direcciones excluidas, la máscara de red, el límite de tiempo que los equipos pueden disfrutar de la concesión, etc.

En cualquier caso, para que un servidor DHCP pueda atender varias redes físicas distintas interconectadas, es necesario que esté conectado a dichas redes, o bien que los encaminadores utilizados tengan la capacidad de encaminar los mensajes del protocolo DHCP entre dichas redes. De no ser así, es necesario utilizar un servidor DHCP distinto en cada red, o bien instalar el servicio de reenvío de DHCP en algún host el cual está configurado para escuchar los mensajes de difusión utilizados por el protocolo DHCP y redirigirlos a un servidor DHCP específico. De esta manera se evita la necesidad de tener que instalar dos servidores DHCP en cada segmento de red.

6.4. Configuración del servidor dhcp.

Existen varias implementaciones del protocolo DHCP para servidores con sistemas operativos parecidos al UNIX, tanto comerciales como libres. Uno de los más populares servidores es DHCPd de Paul Vixie/ISC, que actualmente se encuentra en la versión 3.0, en concreto el paquete que habría que instalar se trata del siguiente: dhcp.

Para configurar DHCPd hay que crear el fichero /etc/dhcpd.conf. Normalmente lo que se pretenderá es asignar direcciones IP de un rango definido de forma dinámica, sin que importe que cliente coge que dirección IP (hay que redordar que DHCP es el protocolo de configuración dinámica de hosts).Esto se podría conseguir con una configuración parecida a la siguiente:

```
default-lease-time 600;
max-lease-time 7200;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.254;
option domain-name-servers 192.168.1.1, 192.168.1.2;
option domain-name "linux.casa";

subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.10 192.168.1.100;
range 192.168.1.150 192.168.1.200;
}
```

En este caso, el servidor DHCP otorgaría una dirección IP al cliente en el rango 192.168.1.10 - 92.168.1.100 o del rango 192.168.1.150 - 192.168.1.200, y concedería la dirección por un tiempo de 600 segundos, mientras el cliente no marque ese tiempo, y en todo caso la concesión máxima será de 7200 segundos. Además configurará el cliente para que use la máscara de red 255.255.255.0, la dirección de difusión 192.168.1.255, el gateway 192.168.1.254, los servidores de nombres 192.168.1.1, 192.168.1.2, y el nombre de dominio linux.casa.

Pero también será necesario que en ocasiones asignemos direcciones de una forma estática (como lo haría el protocolo BOOTP) basándose en la dirección hardware del cliente. El ejemplo siguiente asignará la dirección IP 192.168.1.222 a un cliente con la dirección ethernet 08:00:2b:4c:59:23

```
host haagen {
hardware ethernet 08:00:2b:4c:59:23;
fixed-address 192.168.1.222;
}
```

Si por ejemplo estuvieramos en un entorno mixto de máquinas linux y windows sería a lo mejor necesario asignar a los clientes windows la dirección de un servidor WINS. La opción netbios-name-servers resolvería nuestro problema.

```
option netbios-name-servers 192.168.1.1;
```

Existen muchas más opciones que puede configurar un servidor DHCP. Toda la información se puede encontrar en la página de manual de dhcpd.conf.

6.5. Arranque y parada del servidor dhcp.

Como asi siempre ocurre con todos los servicios de red que no son lanzados por inetd, existe un script en el directorio /etc/init.d que se encarga del asunto,que acepta los parámetros {start|stop} y que deberá estar configurado para que se lance en los niveles de ejecución correctos.

7

Servidores de Correo

Tabla de contenidos

7.1. Introducción.

Este capítulo pretende ser una guía rápida para obtener los conocimientos necesarios a la hora de implantar un sistema de correo electrónico. Se intenta definir el protocolo estandar de correo SMTP y los diferentes agentes que entran en juego en el intercambio de mail.

7.1.1. SMTP: Protocolo de Transferencia de Correo.

El protocolo de transporte de e-mail más utilizado es el protocolo simple de transferencia de correo. Los servidores SMTP, comúnmente denominados MTA's (Mail Transfer Agent), funcionan con un conjunto de reglas limitado:

1. Aceptan un mensaje entrante.
2. Comprueban las direcciones del mensaje.
3. Si son direcciones locales, almacenan el mensaje para después recuperarlo.
4. Si son direcciones remotas, envían el mensaje.

Por tanto, los servidores SMTP son funcionalmente similares a los routers de paquetes. Normalmente, un mensaje pasará a través de varias pasarelas SMTP antes de llegar a su destino final. En cada parada, el servidor SMTP evalúa el mensaje y lo almacena o lo envía. También puede suceder que si el

servidor SMTP encuentra un mensaje que no se puede enviar, este devuelva un mensaje de error al remitente explicando el problema.

7.1.2. MTA's, MDA's y MUA's.

Los MTA's o Agentes de Transferencia de Correo son los encargados de llevar el correo electrónico de un host o red a otro utilizando un protocolo que normalmente es SMTP.

También entran en juego los MDA's (Mail Delivery Agent) o Agentes de Entrega de Correo que se encargan de mover el correo dentro de un sistema, es decir, desde un MTA al buzón local de un usuario, o del buzón a un fichero o directorio.

Además habría que considerar como elementos importantes en el funcionamiento del correo electrónico a los lectores de correo, incluyendo los clientes POP e IMAP que obtienen el correo de un sistema remoto. Estos elementos se les suele conocer como MUA's (Mail User Agent).

Como MTA's más utilizados encontramos sendmail, postfix y qmail. Como MDA's en sistemas UNIX, procmail. Y como MUA's. Elm, pine, Netscape Messanger, OutLook Express.

7.2. Postfix.

Hasta la fecha el servidor de correo más utilizado en Internet venía siendo sendmail (el cual incorpora RedHat por defecto). Postfix proporciona una alternativa a sendmail convirtiéndose en un software de diseño más simple, más modular y más fácil de configurar y administrar.

Sendmail sigue siendo una aplicación muy poderosa, estable y ampliamente desarrollada, pero la curva de aprendizaje y los diferentes agujeros de seguridad encontrados, han hecho que muchos administradores valoren otras alternativas.

7.2.1. Instalación de Postfix.

La forma fácil de instalar postfix es a partir de un paquete rpm con la siguiente línea de comando:

```
$ rpm -ivh postfix-2.0.11-5.i386.rpm
```

Lo único que hay que tener en cuenta es que normalmente las distribuciones vienen con un software MTA instalado por defecto (sendmail) y habría que eliminarlo para que no haya conflictos entre ellos.

```
$rpm -evv sendmail fetcmail mutt
```

7.2.2. Configuración de Postfix.

Los ficheros de configuración de postfix están instalados en el directorio '/etc/postfix' :

```
root@mis21 postfix]# ls install.cf main.cf master.cf postfix-script
```


El fichero principal de configuración para postfix es main.cf. El fichero install.cf contiene valores iniciales que serán definidos durante la instalación del paquete RPM. El fichero master.cf es el fichero de configuración del proceso maestro de Postfix. Cada línea de este fichero describe como un programa componente de correo debe ser ejecutado.

Por tanto administrar Postfix es tan simple como saber moverse a través del fichero main.cf que está además razonablemente comentado por su autor. Este fichero controla alrededor de 100 parámetros de configuración, pero afortunadamente la mayoría de valores por defecto serán una buena elección. Los parámetros son nombres de variable que definen algún comportamiento específico de Postfix (myhostname, mydomain).

En la mayoría de los casos (el nuestro) solo necesitaremos configurar un par de parámetros antes de poder usar el servidor Postfix:

1. Que dominio utilizar para el correo saliente.

   myorigin especificará el dominio que aparecerá en el correo enviado por esta máquina.

2. De que dominios recibimos mail (o lo resolvemos localmente).

   mydestination especifica que dominios resolverá localmente (almacenará) esta máquina en vez de reenviarlo a otra máquina.

Otros parámetros que deberíamos definir en segunda instancia, podrían ser los siguientes:

- myhostname:

  describe el nombre FQDN de la máquina que está ejecutando el servidor Postfix.

- mydomain:

  especifica el dominio padre de la variable $myhostname.

- mynetworks:

  este parámetro lista todas las redes a las que esta máquina está adscrita.

- mydestination:

  este parámetro especifica que dominios tiene que entregar localmente en vez de enviarlo a otras máquinas.

- inet_interfaces:

  especifica todas las direcciones de las interfaces de red donde el sistema Postfix está escuchando.

7.2.3. Controles UCE.

Postfix ofrece una gran variedad de parámetros que limitan la resolución de correo comercial no-solicitado (Unsolicited Commercial Email).

Por defecto, el servidor SMTP Postfix aceptará solamente correo que provenga o vaya a la red local o al dominio, de esta forma el sistema no puede ser usado como un retransmisor de correo (mail relay)

para reenviar desde cualquier usuario extraño.

Sin embargo habrá que adoptar precauciones ya que la línea que separa el spam de los destinos legítimos es muy fina y puede provocar que añadiendo pequeños controles UCE provoquemos que se borre correo legítimo.

Teniendo en cuenta esto, la mayoría de sitios pueden correr el riesgo de definir unos pocos controles como los siguientes:

- smtpd_recipient_limit:

  define cuantos beneficiarios (recipientes) pueden ser direccionados en la cabecera del mensaje.

- smtpd_recipient_restrictions:

  No cualquier correo que llegue al servidor debe ser aceptado. Este parámetro le dice al servidor Postfix que chequee la dirección base del destinatario según algún criterio. Una de las formas mas fáciles es mantener una base de datos de acceso. Este fichero listará dominios, hosts, redes y usuarios que tienen permitido recibir mail a través de tu servidor.

  Para ello habría que habilitarlo de la siguiente forma:

smtpd_recipient_restrictions = check_recipient_acces hash:/etc/postfix/access

- smtpd_sender_restrictions:

  por defecto Postfix aceptará conexiones SMTP de todo el mundo, exponiendo al servidor a lo que se ha dado en conocer como SMTP relaying. Si esto ocurriera, es posible que otros servidores rechacen el correo que provenga de nuestro dominio.

  El mismo fichero de acceso se puede utilizar aquí para prevenir el relay.

7.2.4. Fichero Aliases.

El fichero aliases, normalmente situado en el directorio /etc/postfix, puede ayudar a mapear usuarios que realmente existen en servidores internos, o podemos mapear direcciones de correo de usuarios que no tienen cuenta real en nuestro servidor, o podremos de una forma fácil crear y mantener listas de correo. Una entrada típica en el /etc/postfix/aliases puede ser la siguiente:

Fernando.Ferrer: fferrer@dsic.upv.es [mailto:fferrer@dsic.upv.es]

Una vez que se ha editado el fichero aliases, habrá que generar el nuevo mapa de alias con el comando newaliases.

Si un alias apunta a un servidor de mail diferente, este servidor debe pertenecer a un dominio para el cual el servidor SMTP esté configurado para retransmitir mail. Es decir o bien el FQDN del servidor o su dominio deben de ser incluidos en la variable mydestination.

7.3. Arquitectura de POSTFIX.

Para entender como funciona Postfix, es interesante conocer sus antecedentes. Postfix principlamente surgió debido a la complejidad del otro gran servidor de correo SMTP, sendmail. Postfix, como sabemos, es un MTA que implementa todas las características de este tipo de aplicaciones y ademas sus funciones de nucleo son las mismas que ellas de cualquier otro, pero con la salvedad que postfix fue construido poniendo especial atención a lo ssiguientes puntos:

- Seguridad:

  El sistema postfix no confía en ningún tipo de dato, comprobando siempre su fuente. Y si además lo configuramos para que trabaje en un entorno chroot, el riesgo se reduce.

  Si algo fallara, el mecanismo de protección de Postfix intenta prevenir que cualquiera de los procesos bajo su control ganen derechos que no deberían tener. Ya que el sistema está formado por varios programas que funcionan sin una relación directa entre ellos (a diferencia de un sistema monolítico como sendmail), si algo va mal, el riesgo de que este problema pueda ser explotado por un atacante se minimiza.

- Simplicidad y Compatibilidad:

  Postfix está pensado para ponerlo en marcha desde cero en unos pocos minutos. Además, si el administrador quiere sustituir otro MTA como sendmail, Postfix puede utilizar lo sficheros de configuración antiguos.

- Robusto y Estable:

  Postfix está escrito pensando en que ciertos componentes de la red de correo pueden fallar ocasionalmente. Anticipándose a las cosas que pueden ir mal en una transacción de correo, Postfix es capaz de mantenerse en marcha y corriendo en la maoría de estas circunstancias. Si por ejemplo un mensaje no puede ser entregado, este es programado para ser entregado mas tarde sin iniciar un reintento continuo por enviarlo.

Una gran contribución a la estabilidad y velocidad del servidor Postfix es la forma inteligente en que su creador implementó las colas de correo. Postfix utiliza cuatro colas diferentes, cada una manejada de forma diferente:

1. Maildrop queue:

   El correo que es entregado localmente en el sistema es aceptado por la cola Maildrop. El correo se chequea para formatearlo apropiadamente antes de ser entregado a la cola Incoming.

2. Incoming queue:

   Esta cola recibe correo de otros hosts, clientes o de la cola Maildrop. Mientras sigue llegando correo y Postfix no puede manejarlo, en esta cola se quedan los e-mails.

3. Active queue:

   Es la cola utilizada para entregar los mensajes. La Active queue tien un tamaño limitado, y lo smensajes solamente serán aceptado si hay espacio en ella. Esto quiere decir que las cola Incoming y Deferred tienen que esperar hasta que la cola Active pueda aceptar más mensajes.

7.4. Protocolos IMAP Y POP

El siguiente paso despues de haber instalado un servidor SMTP, es ofrecer al usuario la posibilidad de acceder a su correo. Los clientes de correo (MUA's) serán los encargados de proporcionar esta facilidad

Existen clientes que acceden directamente al buzón del usuario si este tiene la posibilidad de acceder a la maquina que mantiene los buzones (elm, pine, mutt). Pero la alternativa más utilizada es proporcionar al usuario un acceso remoto a su buzón, evitando de esta forma que los usuarios puedan entrar en la maquina que mantiene el servidor de correo.

Los protocolos IMAP y POP permiten al usuario a través de un cliente de mail que los soporte, acceder remotamente a los buzones.

7.4.1. IMAP: Internet Message Access Protocol

IMAP permite a un usuario acceder remotanmente a su correo electrónico como si este fuera local. Además permite leer el correo desde diferentes puestos (clientes) sin la necesidad de transferir los mensajes entre las diferentes máquinas. Por tanto se entiende que los buzones se almacenan en el servidor.

El protocolo incluye funciones de creación, borrado y renombrado de buzones. Es completamente compatible con standars como MIME. Permite el acceso y la administración de mensajes desde más de un ordenador. Soporta accesos concurrentes a buzones compartidos. El cliente de correo no necesita conocer el formato de almacenamiento de los ficheros en el servidor.

7.4.2. POP: Post Office Protocol

POP fue diseñado para soportar el procesamiento de correo fuera de linea. Su funcionamiento se basa en que el cliente de mail se conecta periodicamente a un servidor de correo y se baja (download) todo el correo pendiente a la maquina local del usuario. Por tanto, todo el procesamiento del correo es local a la maquina del usuario, ya que una vez obtenido el correo desde el servidor este es borrado (si el usuario asi lo desea).

POP es un protocolo más simple que IMAP y más fácil de implementar.

7.4.3. Instalación de POP e IMAP

En RedHat la instalación de ambos servicios es una tarea sencilla, ya que en la distribución existe un paquete rpm que soporta ambos protocolos.

La forma de instalación sería la siguiente:

```
root@mis21$ rpm -ivh  imap-2002d-3.i386.rpm
```

7.4.4. Configuración de POP e IMAP

Tanto POP como IMAP son servicios bajo demanda, es decir, el demonio correspondiente se invoca via xinetd cuando se le requiere. Por tanto, la única configuración que tenemos que tener en cuenta,

hace referencia al fichero de configuración del superdemonio xinetd: /etc/xinetd.d/imap|pop.

Este fichero debería las entradas siguientes:

```
service imap
{
 socket_type             = stream
 wait                    = no
 user                    = root
 server                  = /usr/sbin/imapd
 only_from               = 127.0.0.1
 log_on_success  += HOST DURATION USERID
 log_on_failure  += HOST
 disable                 = no
}
```

Como podemos observar el control de acceso a ambos servicios viene definido por el demonio xinetd y como xinetd depende de tcpwrappers el cual utiliza dos ficheros de configuración, /etc/hosts.allow y /etc/hosts.deny , se pueden utilizar para definir los clientes que tienen permitido el acceso a los servicios.

```
#
# hosts.allow   This file describes the names of the hosts which are
#               allowed to use the local INET services, as decided
#               by the '/usr/sbin/tcpd' server.
#
imapd ipop3d : ALL
```

7.5. Otros Recursos

- Diseño de un servicio de Correo electrónico (RedIris) [http://www.rediris.es/mail/estafeta.es.html]
- Manuales sobre Postfix en Redes-Linux [http://redes-linux.dyndns.org/manuales.php]
- RedHat FAQ sobre Postfix [http://www.redhat.com/support/docs/faqs/RH-postfix-FAQ/book1.html]

8

Servidores Web: Apache

Tabla de contenidos

8.1. Introducción.

Con el auge de Internet, muchos son los servicios ofertados a los numerosos clientes que tienen acceso a ella. Entre ellos destaca el correo electrónico o mail y los servidores de Web.

El World Wide Web (Web) es una red de recursos de información. El Web cuenta con tres mecanismos para hacer que estos recursos estén disponibles para el mayor número posible de clientes:

1. Un esquema de nominación uniforme para localizar los recursos en la Web (URI's).
2. La pila de protocolos necesarios para acceder a los recursos definidos a través de la Web (HTTP).
3. El hipertexto para una fácil navegación por los recursos (HTML).

8.1.1. HTTP: Hyper Text Transfer Protocol.

El protocolo de transferencia de hipertexto (HTTP) es un protocolo del nivel de aplicación para sistemas de información hipermedia y distribuidos. Además, es un protocolo orientado a objetos y sin estado.

HTTP viene siendo usado en Internet desde 1990. En este momento la versión de protocolo utilizada es la 1.1.

Con estas palabras comienza el documento RFC2616 que define la especificación del protocolo mas usado en Internet. El protocolo HTTP permite comunicar a ordenadores que sirven información (servidores web) en un determinado formato (HTML: HiperText Markup Language) con ordenadores que consultan dicha información (clientes).

Por supuesto que existe un software especifico para cada función. El software cliente recibe el nombre de navegador (Explorer, Netscape, Amaya, Lynx ...) y el software servidor se denomina también servidor web.

HTTP es un protocolo de petición - respuesta. Un cliente envía una petición al servidor en la forma definida por el método solicitado, una URI y la versión de protocolo, seguido de un mensaje del estilo MIME conteniendo modificadores de petición, información del cliente etc.. El servidor responde con una línea de estado que incluye la confirmación de la versión del protocolo y un código de error o de éxito seguido por información del servidor y la información solicitada, terminándose acto seguido la comunicación.

8.1.2. URI: Uniform Resource Identifiers.

La forma de acceder a los recursos que ofrecen los servidores Web, es especificando en el navegador una URI (Identificador Uniforme de Recursos).

Para el protocolo HTTP un URI es un string formateado que identifica por medio de un nombre, o una localización, un recurso en la red. Una URI bajo el punto de vista del protocolo HTTP puede ser representada de forma absoluta o relativa, dependiendo del contexto en donde se la use.

Ambas formas se diferencian en el hecho de que las URI's absolutas empiezan siempre por un nombre de protocolo seguido por dos puntos ':'.

Básicamente las URI's constan de tres partes:

1. El esquema de nominación del mecanismo utilizado para acceder al recurso.
2. El nombre de la máquina que alberga el recurso.
3. El nombre del recurso propiamente dicho, dado como un path.

```
http : // host [ : puerto ] [ path absoluto ] [ ? consulta ]
```

Si el puerto no se especifica, se asume el puerto 80 por defecto.

8.1.3. HTML: HyperText Markup Language.

HTML es una aplicación SGML (Standard Generalized Markup Language) conforme al standard internacional ISO 8879 y es reconocido como el lenguaje de publicación estándar en el World Wide Web.

SGML es un lenguaje para describir lenguajes de marcas, utilizados particularmente en el intercambio de información electrónica, gestión de documentos y publicación de los mismos. HTML es un ejemplo de lenguaje definido en SGML.

HTML fue originariamente concebido como un lenguaje de intercambio de documentos científicos y técnicos por Tim Berners-Lee mientras trabajaba en el CERN y popularizado por el navegador Mo-

saic desarrollado en NCSA.

HTML proporciona los medios para:

- Publicar online documentos con cabeceras, texto, tablas, listas, fotos etc ...
- Obtener información en línea vía enlaces de hipertexto con un solo clic del ratón.
- Diseñar formularios para realizar transacciones con servicios remotos, que nos permitan búsqueda de información, realizar reservas, comprar productos.
- Incluir hojas de cálculo, video-clips, sonidos y otras aplicaciones directamente en los documentos.

8.2. Servidores WEB.

A la hora de instalar un servidor Web debe tenerse en cuenta una serie de consideraciones previas que tienen que ver con el soporte físico (hardware) sobre el que correrá el servidor: interfaces de red, sistema de almacenamiento SCSI con soporte RAID, memoria RAM de al menos 256 MB, procesador dependiente de si el contenido del sitio Web es mas bien dinámico o estático, y sobre todo si tiene que acceder a diferentes bases de datos.

Y en cuanto al software, el sistema operativo será también una elección importante a la hora de montar un sitio Web, junto con el software servidor de Web.

El software servidor de Web elegido para las prácticas es el servidor de libre distribución Apache. No solo porque sea gratuito y dispongamos del código fuente, sino porque actualmente como indican las estadísticas de Netcraft, el servidor de Web Apache ocupa un 62.51 % del mercado.

Apache empezó como una serie de parches al servidor de Web desarrollado en el National Center for Supercomputing Application (NCSA) y una vez abandonado el proyecto de NCSA, programadores de todo el mundo encontraron la necesidad de tener un repositorio central donde mantener el código y los parches del nuevo software. Así surgió la Apache Software Foundation.

Apache fue diseñado desde el principio de forma modular, así los programadores originales asumen que el software puede ser ampliado por otros desarrolladores, los cuales pueden escribir pequeñas partes del código que se integrará en Apache de una manera fácil. Esto se lleva a cabo al haber creado un API modular y una serie de fases bien definidas por las que cada petición al servidor debe atravesar. Estas fases van desde la inicialización del servidor (cuando Apache lee los ficheros de configuración), hasta la traducción de una URL en un nombre de fichero del servidor, o registrar los resultados de la transacción.

Por contra, esta modularidad de Apache puede hacer potencialmente difícil la configuración del servidor. Por defecto, Apache viene con un buen número de módulos habilitados por defecto. Si uno se encuentra en disposición de poder compilar el código fuente, puede adecuar el servidor a sus necesidades obteniendo unos resultados inmejorables.

8.3. Instalación de Apache Web Server.

No vamos a considerar en este seminario la posibilidad de compilar el propio servidor, sino que partiremos de la base de que disponemos de una distribución binaria del servidor compilada para nuestra plataforma.

- Instalación en Windows 2000 consiste tan solo en ejecutar el programa de autoinstalación, el cual

nos solicitara el directorio de instalación, el nombre de la entrada en el menú de inicio, y que tipo de instalación necesitamos.

- Instalación en Fedora Core Linux 1

  En un sistema Linux que soporte el formato de paquetes RPM, la instalación es tan sencilla como invocar al comando rpm o utilizar alguna herramienta gráfica como gnorpm o kpackage

  $ rpm -ivh apache-1.3.29-2.i386.rpm

  Este comando hará el resto del trabajo dejando el software en el sitio apropiado.

- Instalación desde los fuentes

  $ tar zxvf apache-1.3.29.tar.gz

  $ cd apache-1.3.29

  $ ./configure --prefix=/usr/local/apache

  $ make; make install

Se escoja el sistema operativo que se escoja, existen unos directorios importantes a la hora de trabajar con el servidor de Web Apache.

- Icons: imágenes que utiliza Apache.
- Htdocs: documentos HTML.
- Cgi-bin: programas CGI.
- Conf: ficheros de configuración del servidor.
- Logs: archivos de registro de acceso y error.

8.4. Configuración de Apache.

El servidor lee la información de tres ficheros con directivas de configuración. Cualquier directiva puede aparecer en cualquiera de estos ficheros. Los nombres de estos ficheros son relativos a la raíz de instalación del servidor, la cual es definida por la directiva ServerRoot, la opción en línea de comando -d o en un sistema Windows por el registro:

- conf/httpd.conf:

  Contiene las directivas que controlan el funcionamiento del demonio servidor (httpd).

- conf/srm.conf:

  contiene las directivas que controlan la especificación de documentos que el servidor proporciona a los clientes.

- conf/access.conf:

  Contiene directivas que controlan el acceso a los documentos

Además, el servidor también leerá un fichero conteniendo los tipos mime de los documentos. Por defecto el fichero será conf/mime.types.

Sin embargo, la tendencia es que solo se utilice como fichero principal del servidor Web a conf/httpd.conf, agrupando las directivas de configuración en tres secciones:

8.4.1. Global Enviroment.

En esta sección se definen las directivas que controlan el funcionamiento general de Apache, como pueden ser el número de peticiones concurrentes que puede manejar, donde puede encontrar los ficheros de configuración...

- ServerType { standalone | inetd }:

  El servidor se lanza bien como un servicio independiente que está corriendo como un demonio esperando alguna petición (standalone) o bien es lanzado por el superservidor de internet (inetd) como un servicio bajo demanda.

- ServerRoot:

  Define el directorio donde reside toda la información de configuración y registro que necesita el servidor.

- PidFile:

  Indica el fichero en el cual el servidor registrará el identificador de proceso con el que se lanza.

- ScoreBoardFile:

  Fichero utilizado para almacenar información interna del proceso servidor. Se requiere para algunas arquitecturas para comunicar entre los procesos hijos y el padre.

- Timeout:

  Número de segundos tras los cuales el servidor cierra la conexión.

- KeepAlive:

  Si se permiten o no conexiones persistentes.

- MaxKeepAliveRequest:

  El número máximo de peticiones permitidas durante una conexión persistente.

- MinSpareServers:

  Define el número de procesos servidores hijo desocupados que no atienden una petición.

- MaxSpareServers:

  Define el nº máximo de procesos servidor hijo desocupados que no manejan una petición. Si existieran mas de lo que define la directiva, el proceso padre mataría los procesos que exceden.

- StartServers:

  Número de procesos servidor hijo que serán creados cuando arranca Apache por primera vez.

- Maxclients:

  Define el número de peticiones simultáneas que apache puede soportar. Como máximo se crean este nº de procesos servidores hijo.

- MaxRequestPerChild:

  Define el nº de peticiones que cada proceso hijo tiene permitido procesar antes de morir.

- ThreadsPerChild:

  Nº de hilos concurrentes que el servidor permitirá utilizar. Este valor representa el nº máximo de conexiones que el servidor puede manejar a la vez.

- Listen:

  Esta directiva instruye al servidor Apache para que escuche en mas de una dirección IP o en mas de un puerto.

- LoadModule:

  Esta directiva enlaza dentro del servidor la librería o fichero objeto nombrado cuando se arranca el servidor y añade la estructura del modulo a la lista de módulos activos.

8.4.2. Main Server Configuration.

Estas directivas definen los parámetros del servidor principal, el cual responde a las peticiones que no son manejadas por un host virtual. También proporciona parámetros por defecto para todos los hosts virtuales.

- Port:

  Define el puerto en el cual escucha el servidor (0 - 65535). Hay que tener en cuenta la relación que tiene esta directiva con el fichero /etc/services y que algunos puertos, especialmente los situados por debajo del 1024, están reservados para protocolos específicos. El puerto estándar para el protocolo HTTP es el 80.

- User/Group:

  Definen el usuario y el grupo con el que el servidor contestará las peticiones. Para poder utilizar esta directiva, el servidor standalone debe ejecutarse inicialmente como usuario root. El usuario no debería poseer privilegios que otorguen acceso a ficheros que no deseemos.

- ServerAdmin:

  Define la dirección de correo que el servidor incluirá en cualquier mensaje de error que devuelva al cliente.

- ServerName:

  Define el nombre de host del servidor. Se suele utilizar cuando se crean redirecciones. Sino se define el nombre de servidor, intentará deducirlo a través de su dirección IP.

- DocumentRoot:

  Define el directorio desde el cual el servidor servirá los documentos. A menos que la URL solici-

tada coincida con una directiva Alias, el servidor añadirá el PATH a la URL.

- Directory:

  <Directory></Directory> se utilizan para encerrar un grupo de directivas que se aplicarán al directorio en cuestión y sus sub-directorios. El parámetro directorio, puede ser una trayectoria completa o un metacaracter.

- Options: [+|-] opcion:

  Controla que características están disponibles para un directorio en particular. La opción se puede definir a None, de forma que ninguna característica extra se habilita o a All de forma que todas se habilitan menos MultiViews. Otras características extra son: ExecCGI , FollowSymLinks, Includes, IncludesNOEXEC, Indexes, MultiViews, SymLinksIfOwnerMatch.

- AllowOverride override:

  Cuando el servidor encuentra un fichero .htaccess (definido por la directiva AccessFileName) necesita conocer que directivas declaradas en este fichero pueden sobreescribir información de acceso.

  El parámetro override puede ser definido a None y en tal caso el servidor no leerá el fichero o puede ser definido a All, de forma que permitirá todas las directivas.

  El parámetro override también puede ser definido a: AuthConfig, FileInfo, Indexes, Limit, Options.

- UserDir directorio:

  Define el nombre del directorio que será añadido al directorio HOME del usuario si se recibe una petición ~usuario.

- DirectoyIndex fichero:

  Nombre/s del fichero/s a usar como página de inicio de un directorio.

- AccessFileName fichero:

  El nombre del fichero a buscar en cada directorio para información de control de acceso.

- Alias url directorio:

  Esta directiva permite que los documentos sean almacenados en un sistema de ficheros diferente al definido por la directiva DocumentRoot.

- Location url:

  La directiva proporciona control de acceso por URL. Es similar a la directiva Directory.

- ScriptAlias url directorio:

  Tiene el mismo comportamiento que la directiva Alias, excepto que además define el directorio para que pueda contener scripts CGI.

8.4.3. Virtual Hosts.

Un servidor Web puede albergar diferentes sitios web utilizando diferentes nombres de hosts (alias) o diferentes IP's. En esta sección se define el comportamiento específico de cada servidor virtual que alberga el servidor.

- <VirtualHost> </VirtualHost>

  <VirtualHost> and </VirtualHost> se utilizan para agrupar un conjunto de directivas que se aplicarán solo a un host virtual. Dentro de este par de directivas se pueden incluir cualquier otra que haga refenrencia a este host particular.

- NameVirtualHost:

  Especifica la dirección a resolver cuando se utiliza un sistema basado en nombres de host virtuales.

- ServerName:

  Especifica el nombre de host que aparecerá en la cabecera de la petición.

Los beneficios de utilizar el soporte de host virtuales permite definir prácticamente un número de servidores ilimitado, de fácil configuración y que no requiere hardware adicional. La principal desventaja es que el software cliente debe soportar esta parte del protocolo.

Pero como veremos, es fácil mantener varios sitios web con diferentes nombres de dominio tan solo con definir varios alias (CNAME) para ese host:

NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>

ServerName www.domain.tld

DocumentRoot /www/domain

</VirtualHost>

<VirtualHost 111.22.33.44>

ServerName www.otherdomain.tld

DocumentRoot /www/otherdomain

</VirtualHost>

8.5. Autenticación y autorización.

La autentificación se basa en el principio de que el cliente envia su nombre y su password al servidor, un modulo de Apache chequea si las credenciales son correctas, y si lo son devuelve la página solicitada. Si el usuario no tiene permitido el acceso o el password no es válido, el servidor Apache devuelve un código de estado 401 (Acceso no autorizado).

8.5.1. Acceso Restringido por Usuario y Password.

Para poder definir este tipo de autenticación, es necesario seguir dos pasos:

1. Crear un fichero que contenga los usuarios y los passwords.

   Htpasswd -c -b /etc/httpd/conf/db_users fferrer MIS2000

2. Definir las zonas a restringir.

   Lo cual se consigue definiendo un fichero llamado .htaccess en el directorio que queremos restringir.

8.5.2. .htaccess: Directivas de Configuración.

Cuando el servidor tiene que acceder a una zona donde se encuentra un fichero .htaccess (definido por AccessFileName) comprobará las directivas definidas en el para otorgar o no el acceso.

El servidor además necesita conocer que directivas declaradas en este fichero pueden sobreescribir la información de acceso anterior declarada por la directiva AllowOverride.

- AuthName:

  Define un nombre para la zona protegida. Una vez que un usuario introduzca unas credenciales válidas para esta zona, cualquier otro recurso dentro de esta zona podrá ser accedido con las mismas credenciales.

- AuthType:

  Define el sistema de autentificación utilizado. Pede ser Basic o Digest. Este último método ofrece mayores características de segurudad al utilizar firmas MD5.

- AuthUserFile:

  Define la localización del fichero creado con el comando httpasswd.

- Require:

  Define que nombres de usuario del fichero definido por la directiva anterior tienen derechos de acceso. Valid-user sería un caso especial refiriendose a cualquier usuario del fichero.

.htacces

AuthName Protegido

AuthType Basic

AuthUserFile /etc/httpd/conf/db_usersrequire

valid-user

9

Seguridad en red en GNU/Linux

Tabla de contenidos

9.1. Introducción.

Ya se vio como proteger a nivel local un sistema GNU/Linux, en cuanto a accesos locales por parte de usuarios. RedHat Linux a la hora de crear usuarios se decantaba por una estrategia conocida como grupos privados. De esta forma solo el usuario era capaz de acceder a su información y si éramos algo paranoicos podíamos obligar al usuario a que no cambiara los permisos de su directorio particular.

En este tema, el objetivo es proteger nuestro sistema ante accesos no autorizados a través de la red. Si nuestro servidor GNU/Linux está conectado a una Red de Area Local o directamente a Internet, podemos encontrarnos con desagradables situaciones sino hemos tomado las mediadas necesarias.

Por tanto uno de las principales tareas de un administrador será saber que servicios debe de ofrecer su sistema y a quien se los puede ofrecer. GNU/Linux como sistema operativo orientado a la Red, incorpora gran número de herramientas que nos permitirán proteger nuestro sitio.

9.2. Servicios.

Para ver los servicios que ofrece nuestra máquina se puede usar el comando netstat. Por ejemplo:

```
# netstat -atu | grep '*:*' | more
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 *:ftp *:* LISTEN
tcp 0 0 *:telnet *:* LISTEN
tcp 0 0 *:printer *:* LISTEN
tcp 0 0 *:6000 *:* LISTEN
udp 0 0 *:syslog *:*
```

En un sistema Linux con una instalación por defecto, esta lista será mucho más larga. Nuestro objetivo es hacerla lo más pequeña posible, pero de forma que cubra nuestras necesidades.

Los servicios más comunes suelen tener asociado un nombre, que aparece en la columna etiquetada como "Local Address". En el caso de que el servicio no tenga asociado un nombre, en esta columna aparecerá el número de puerto en el que el servidor se encuentra a la escucha. En este ejemplo aparece un servicio en el puerto 6000, que corresponde al servidor de X.

Durante la instalación del sistema operativo se puede seleccionar qué servidores deseamos instalar en nuestra máquina. Algunos de ellos se arrancan en el momento del arranque inicial, mientras que otros son ejecutados bajo demanda, es decir, el proceso sólo se ejecuta cuando se establece una petición de servicio.

9.2.1. Servicios de Arranque directo.

El mismo programa de instalación pregunta qué servicios se desea arrancar durante el inicio de la máquina.

Entre estos servicios se encuentran:

- *Servicio de impresión*: si quieres utilizar una impresora desde tu estación tendrás que dejar este servicio activo.
- *Syslog*: no lo quites, es el servicio encargado de registrar los eventos que ocurren en tu máquina (por ejemplo, los intentos de conexión).
- *NFS*: permite exportar directorios o discos a otros sistemas Unix. Aunque en principio esto puede parecer bastante deseable, con frecuencia NFS es fuente de problemas de seguridad, así que si se instala es conveniente revisar su configuración y estar pendiente de los parches de seguridad que vayan saliendo. A menos que realmente haga falta, nosotros recomendamos desactivarlo.
- *Samba*: permite exportar directorios o discos a sistemas bajo Windows. Al igual que NFS, suele estar "en el punto de mira". A menos que realmente te simplifique la vida, no lo uses.
- *Correo SMTP*: el programa que hace de servidor es difícil de configurar y tiene un amplio historial de agujeros de seguridad. Si dispones de una cuenta POP, te resultará mucho más sencillo utilizar un cliente de correo compatible (por ejemplo, pine para modo texto o netscape para X). Más adelante os detallamos un poco más sobre el sendmail.
- *Web, news, DNS*: cualquiera de estos servicios requiere una configuración previa que puede ayudar a restringir el acceso a los mismos. Para una estación de trabajo, no hacen falta y es mejor no instalarlos.
- *Otros servicios*: los siguientes servicios son muy específicos, y es bastante improbable que los necesites.

```
bootparamd
 dhcpd
 gated
 routed
 rusersd
 rwalld
 rwhod
 snmpd
 squid
 xntpd
 ypbind
 yppasswdd
 ypserv
```

La forma mas directa y fácil de eliminar estos servicios es utilizar alguna herramienta de administración como el control-panel o linuxconf, o en modo texto el setup.

Desde la linea de comandos siempre podremos invocar a chkconfig para que elimine los enlaces que se generan en los diferentes directorios que controlan los niveles de ejecución.

9.2.2. Servicios de arranque bajo demanda.

Los servicios de arranque bajo demanda son controlados en sistemas UNIX por el demonio inetd y aparecen listados en el fichero /etc/inetd.conf. Cuando inetd recibe una petición por un puerto asociado a un servicio lanza en ese momento el servicio correspondiente pasando a la escucha de nuevo.

Actualmente en RedHat, esta función la desempeña xinetd (Extended Internet Services Daemon). Los servicios más habituales que controla este super-daemon pueden ser los siguientes:

- *ftp*: permite transferir ficheros hacia o desde nuestra estación de trabajo. Recomendamos permitir el acceso por FTP sólo desde un rango limitado de máquinas.
- *telnet*: permite abrir una sesión remota en nuestra estación. También es recomendable limitar el acceso por máquinas.
- *shell (rsh), login (rlogin)*: es preferible quitarlos, siempre que sea posible, puesto que permiten entrar en la máquina o ejecutar comandos de forma remota sin necesidad de dar un password. Si se dejan, limitar siempre el acceso por máquinas. Otra posibilidad es utilizar un sustituto más seguro como el SSH (Secure Shell) que se discutirá en un punto posterior..
- *Gopher*: se trata de un servicio de información poco usado en la actualidad; los detalles de su configuración quedan fuera del ámbito de esta guía.
- *talk, ntalk*: permiten mantener una conversación interactiva con un usuario en la misma máquina o en otra remota.
- *pop-2*: es una versión antigua del protocolo POP; se puede quitar.
- *pop-3*: es el servidor del protocolo de correo POP3; si dispones de un a cuenta POP y simplemente quieres leer tu correo, este servicio NO hace falta para nada.
- *imap*: otro protocolo de correo, similar al POP; tampoco hace falta en una estación de trabajo.
- *finger*: puesto que puede dar información útil para un atacante (por ejemplo, si hay alguien conectado que pueda darse cuenta de una entrada anómala), la mayoría de los administradores suelen quitarlo.
- time: sirve para sincronizar el reloj entre un grupo de máquinas; lo mejor es quitarlo.
- *auth*: mantiene un registro de qué usuario está ejecutando qué servicio TCP. Puede ser útil para el administrador.

La configuración del super-daemon xinetd viene organizada de la siguiente forma:

- Fichero */etc/xinietd.conf*: contiene los parámetros de configuración global, como pueden ser los parámetros relativos al registro de los accesos en los ficheros del sistema.
- Directorio */etc/xinetd.d*: aquí se alojan los diferentes archivos de configuración de cada servicio controlado por xinetd. En cada uno de ellos se específican los parámetros de configuración indivi-

duales de cada daemon.

El fichero de configuración de xinetd por defecto que trae RedHat es el siguiente:

```
defaults
{
        instances               = 60
        log_type                = SYSLOG authpriv
        log_on_success          = HOST PID
        log_on_failure          = HOST
}
includedir /etc/xinetd.d
```

Donde cada opción especifica lo siguiente:

- instance=60: especifica el nº máximo de peticiones que pueden estar simultáneamente activas para un servicio.
- log_type= SYSLOG authpriv: la opción especifica el formato que queremos usar para capturar la salida de un servicio. Los valores son FILE o SYSLOG.
- log_on_succes = HOST PID: esta opción especifica que información será registrada cuando se inicie el servicio. Los valores que acepta son los siguientes: PID HOST USERID EXIT DURATION
- log_on_failure = HOST: especifica que será registrado bien cuando el servicio no pueda ser iniciado debido a falta de recursos o porque se ha denegado el acceso según las reglas establecidas. Valores: HOST USERID ATTEMPT RECORD.
- Includedir /etc/xinetd.conf: define el directorio donde residirán los ficheros de configuración de cada servicio.

9.3. Tcpwrapper.

Sería útil poder restringir los servicios, tanto aquellos que se ejecutan bajo demanda como los que lanzamos directamente en el proceso de arranque, dependiendo de la máquina (dirección IP) que quiere optar al servicio. Esta funcionalidad se puede conseguir si instalamos el paquete tcp_wrappers.

Los ficheros en los que se restringe el acceso son */etc/hosts.allow* y el fichero */etc/hosts.deny*.

La configuración más normal es denegar todo lo que no esté permitido en el /etc/hosts.allow.

Por ejemplo, si sólo queremos permitir el acceso por telnet , rlogin y pop a la máquina xxx.xxx.xxx.xxx, y a todo nuestro dominio linux salvo la máquina mio.linux, y el resto denegarlo, la configuración de estos dos ficheros sería:

En el /etc/hosts.allow:

```
in.telnetd: xxx.xxx.xxx.xxx .cica.es EXCEPT mio.linux
in.rlogind: .cica.es xxx.xxx.xxx.xxx EXCEPT mio.linux
ipop3d: .cica.es xxx.xxx.xxx.xxx EXCEPT mio.linux
```

y en el /etc/hosts.deny tener la siguiente línea:


```
ALL:ALL
```

En el caso de que decidamos utilizar inetd, el wrapper actúa de intermediario entre el servidor inetd y los servicios controlados por inetd definidos en /etc/inetd.conf (telnet, ftp,...), permitiendo el acceso sólo a las máquinas que configuremos, y por supuesto haciendo registro de todos los accesos a estos servicios. También serán susceptibles de configurarse via el tcpwrapper aquellos servicios que siendo demonios hayan sido compilados con soporte tcpwrapper. Por ejemplo el portmap o el ypserv, pueden utilizarlo.

Para configurarlo, basta sustituir en el fichero /etc/inetd.conf, cada servicio por el demonio tcpd. Por ejemplo, si queremos restringir el acceso al servicio telnet, en lugar de esta línea:

```
telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
```

aparecerá esta otra:

```
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

Lo que hay que resaltar que aunque actualmente en RedHat los servicios bajo demanda los controla xinetd y este puede hacer la misma función que el wrapper, existen daemons que pueden ser compilados con la librería tcpwrapper y utilizar los ficheros de configuración de este. Por ejemplo, este es el caso de SSH.

9.4. Secure Shell (SSH).

SSH (*Secure Shell*) es un programa de login remoto que permite una transmisión segura de cualquier tipo de datos: passwords, sesión de login, ficheros, sesión X remota, comandos de administración, etc. Su seguridad estriba en el uso de criptografía fuerte de manera que toda la comunicación es cifrada y autentificada de forma transparente para el usuario. Es un claro y sencillo sustituto de los típicos comandos "r" de BSD (rlogin, rsh, rcp), telnet, ftp e incluso de cualquier conexión TCP.

Entre las mejoras que se pueden apreciar en el uso de SSH se encuentra una autenticación más robusta de usuarios y hosts, que la tradicionalmente ofrecida basada en direcciones IP y nombres de máquinas. Una privacidad mayor para el usuario debido al uso de canales de encriptación.

Así, el ssh establece un entorno protegido contra ataques típicos como: Ip Spoofing, Ip source routing, DNS spoofing, Sniffing, X11 Server Spoofing, etc.

9.4.1. Autenticación por password.

SSH permite autentificar a un usuario utilizando su password Unix ordinario. La única (e importante) diferencia es que el password no viaja nunca en claro por la red. Si utilizamos SSH para sustituir a telnet, rlogin o ftp evitaremos el peligro de que nuestro password sea capturado por posibles "sniffers" en la red.

Por otra parte, seguiremos siendo vulnerables a los llamados "ataques de diccionario" contra el password: si un atacante tiene acceso al fichero /etc/passwd, no resulta difícil averiguar passwords formados a partir de palabras susceptibles de figurar en un diccionario. Esto significa que sigue siendo extremadamente importante que el administrador proteja debidamente el fichero /etc/passwd y que los usuarios utilicen passwords "seguros" (lo más aleatorios posible, combinando mayúsculas, minúsculas, dígitos y puntuación).

La forma de uso de ssh es muy similar a la de rsh:

```
% ssh remoto
Accepting host remoto key without checking.
usuario's password:
```

Este comando nos permitiría iniciar una sesión interactiva en el nodo remoto, tras registrar la clave de servidor del mismo.

También se puede utilizar ssh para ejecutar un único comando en el servidor:

```
% ssh remoto pwd
Accepting host remoto key without checking.
usuario's password:
/home/usuario
```

A diferencia del comando rsh, ssh siempre pide que introduzcamos nuestro password. Además, muchas de las opciones de rsh, si no todas, están duplicadas para el comando ssh. Por ejemplo, si queremos conectarnos a una cuenta remota con un nombre de usuario diferente usaremos la opción "-l":

```
% ssh remoto -l otro_usu
```

Cada servidor de SSH tiene asociado un par de claves pública/privada, equivalentes al par de claves de un usuario. Estas claves se utilizan para identificar al servidor frente al usuario; de esta forma, se evita que otra máquina (posiblemente hostil) pueda suplantarlo.

La primera vez que nos conectamos a la máquina remota, SSH solicita al servidor su clave pública, y la acepta sin hacer ninguna comprobación. Esta clave queda registrada en el directorio $HOME/.ssh2/hostkeys del usuario; en futuras conexiones el cliente comparará la clave que le suministra el servidor con la que tiene almacenada. Si ambas claves no coinciden, el cliente no establecerá la conexión, para evitar que alguna máquina hostil pueda suplantar impunemente al servidor. Si en algún momento la clave del servidor cambia por un motivo legítimo (por ejemplo, por una reinstalación del sistema operativo), tendremos que borrar el fichero que contiene la clave del nodo remoto de nuestro directorio $HOME/.ssh2/hostkeys.

Hay que tener en cuenta que a la hora de almacenar las claves de los servidores, SSH no hace traducción de alias a nombres. Por ejemplo, "remoto.dominio.es" y "remoto" son tratados por SSH como dos servidores distintos, cada uno con su clave de host propia, aunque ambos nombres se traduzcan a la misma dirección IP.

9.4.2. Autenticación por clave pública.

La segunda alternativa de autentificación utiliza un esquema de clave pública/privada, también conocido como clave asimétrica. En este esquema se utiliza un par de claves:

1. una clave pública, que se copia a todos los servidores a los que queremos conectarnos.
2. una clave privada, que solamente nosotros poseemos; para mayor seguridad, esta clave está cifrada con una frase de paso.

Estas dos claves poseen una característica importante: un texto cifrado con la clave pública sólo puede ser descifrado usando la clave privada, mientras que un texto cifrado con la clave privada sólo puede

descifrarse mediante la clave pública.

Veamos cómo se aplica esta propiedad al proceso de autentificación:

1. el servidor nos envía un mensaje, que debemos devolver cifrado con nuestra clave privada.
2. el servidor descifra el mensaje de respuesta con nuestra clave pública.
3. el servidor compara el mensaje resultante con el texto original; si coinciden, el servidor nos considera debidamente autentificados

Por supuesto, todo este proceso es transparente para el usuario; nosotros solamente tendremos que preocuparnos de teclear nuestra frase de paso cuando el programa nos lo pida. El punto más débil de este esquema es cómo hacer llegar nuestra clave pública al servidor. De momento no existe ninguna forma automática de hacerlo, y no hay más remedio que hacerlo a mano.

La principal ventaja de este método de autentificación es que, aunque un atacante lograra comprometer el servidor, sólo podría conseguir acceso a nuestra clave pública, pero nunca a nuestra clave privada. De todas formas, en prevención de un posible compromiso del cliente, es necesario que la clave privada esté protegida con una frase de paso adecuada. De esta forma, nadie podrá utilizarla aún en el caso de que consiguiera de alguna manera hacerse con ella.

En el caso de realizar la conexión desde una máquina Unix, una ventaja secundaria es que se puede utilizar un agente de autentificación para evitar tener que teclear la frase de paso en cada conexión.

El mayor inconveniente de la autentificación por clave pública es la fase de configuración previa, que puede resultar algo engorrosa. Los pasos a seguir son:

1. generación de las claves.
2. propagación de la clave pública.
3. selección del par de claves

9.4.3. Transferencia de ficheros.

Para la transferencia de ficheros existen dos alternativas:

El comando *scp*, equivalente al comando rcp tradicional; resulta muy útil para transferencias simples desde la línea de comandos. La forma de uso imita a la de rcp, con opciones similares. Por ejemplo:

```
% scp fichero remoto:
% scp remoto:/path/de/fichero /path/local
% scp fichero1 fichero2 remoto:dir
% scp fichero otro_usu@remoto:
```

El comando *sftp*, que intenta emular la forma de uso de un cliente FTP ordinario. Es útil para transferencias más complicadas, por ejemplo si tenemos que movernos por la estructura de directorios o listar su contenido. De todas formas, el número de comandos soportados es limitado.

```
% sftp remoto
local path  : /home/usuario
remote path : /home/usuario
```

```
sftp>help
?           dir         ldelete     mkdir       quit        user
bell        disconnect  ldir        open        recv        verbose
bye         get         lls         page        rm
cd          hash        lpwd        prompt      sort
close       help        lrm         put         status
delete      lcd         ls          pwd         timeout

? command  for more information on a specific command.
sftp>
```

10

Tutorial del lenguaje Python

Tabla de contenidos

10.1. "Hello World!"

El primer programa al que se enfrentará el desarrollador más avanzado cuando empieza con un nuevo lenguaje, será sin ninguna duda el famoso ¡ Hola Mundo !

```
>>> print 'Hello World!'
Hello World!
```

La sentencia **print** se utiliza en python para mostrar la salida por la pantalla. Aquellos que estén familiarizados con C, deben saber que funciona de forma parecida a la sentencia **printf()**.

Un apunte final, la sentencia **print** junto con el operador de formateo de cadenas (%) se comporta exactamente igual a la sentencia **printf()** de C.

```
>>> print "%s es el número %d!" % ("Python", 1)
Python es el número 1!
```

10.2. Entrada de Usuario. raw_input()

La forma más fácil de obtener la entrada de datos de un usuario por pantalla, es utilizando la sentencia **raw_input()**.

```
>>> user = raw_input('Introduce tu login: ')
Introduce tu login: root
>>> print 'Tu login es:', user
Tu login es: root
```

El ejemplo anterior espera que la entrada sea una cadena, pero si queremos introducir un valor numérico y que sea tratado como tal, habrá que convertirlo previamente.

```
>>> num = raw_input('Introduce un número: ')
Introduce un número: 1024
>>> print 'El doble del número es: %d' % (int(num) * 2)
El doble del número es: 2048
```

La función **int()** convierte la variable de tipo *string* en un entero.

10.3. Operadores

- **Operadores matemáticos**: + - * / % **
- **Operadores de comparación**: < <= > >= == != <>
- **Operadores lógicos**: and or not

10.4. Variables

Las reglas que rigen el comportamiento de las variables en Python son muy parecidas a las de otros lenguajes. Las variables son simples identificadores que deben empezar siembre por un carácter alfabético, en mayúsculas o minúsculas o el símbolo _ (guión bajo), seguidos de cualquier carácter alfanumérico.

Hay que recalcar que Python distingue entre mayúsculas y minúsculas.

Python es un lenguaje de tipos dinámicos, lo cual significa que no hay necesidad de declarar el tipo de una variable. El tipo se definirá en el momento de la asignación de la variable.

```
>>> contador = 0
>>> millas = 1000.0
>>> nombre = 'Fernando'
>>> contador = contador + 1
>>> kms = 1.609 * millas
>>> print '%f millas es lo mismo que %f km' % (millas, kms)
1000.000000 millas es lo mismo que 1609.000000 km
```

10.5. Números

Python soporta cuatro tipos numéricos:

1. Enteros con signo (*int*).
2. Enteros largos (*long*) que pueden ser representados también en octal o hexadecimal.
3. Reales en coma flotante (*float*)
4. Números complejos (*complex*)

Los tipos realmente interesantes en Python por las particularidades que comportan son el tipo *long* y el tipo *complex*.

El tipo long es superior al archiconocido long de C, ya que en Python no tiene límite de capacidad, solamente el que le imponga la memoria virtual del sistema. Sería más parecido a los números definidos en Java con la clase BigInteger.

En cuanto al soporte de números complejos, Python es el único lenguaje que nativamente soporta este tipo de datos.

10.6. Secuencias (*strings, listas y tuplas*)

Las secuencias son un tipo de datos cuyos elementos están ordenados y pueden ser accedidos vía un índice.

Todos los tipos de secuencias comparten el mismo modelo de acceso a sus elementos. Para acceder a un elemento en concreto se utiliza la siguiente nomenclatura *seq[i]*. Dada una variable de tipo secuencia denominada seq, accedemos al elemento que ocupa la posicion i. El esquema de numeración utilizado empieza en el 0 y finaliza en el valor que defina la longitud de la secuencia menos 1.

Varios elementos (substrings) pueden ser obtenidos a la vez utilizando el operador slice (trozo). La sintaxis para obtener un grupo de elementos es la siguiente:

```
secuencia[indice_inicial : indice_final]
```

Con esta sintaxis podemos obtener un trozo (slice) empezando en el elemento definido por el indice_inicial y terminando en el elemento anterior al definido por el indice_final.

```
>>> cadena='Hola Mundo!'
>>> print cadena[0:4]
Hola
```

Parece algo confuso pero la mejor forma de recordar como funciona el operador slice es pensar que los indices apuntan realmente entre los caracteres.

```
 +---+---+---+---+---+
 | H | e | l | p | A |
 +---+---+---+---+---+
 0   1   2   3   4   5
-5  -4  -3  -2  -1
```

En la siguiente tabla se muestra una lista de operadores que se pueden utilizar con todos los tipos de secuencias:

Tabla 10.1. Operadores de secuencias

secuencia[index]	elemento situado en el índice index de la secuencia
secuencia[ind1:ind2]	elementos desde el índice ind1 hasta el índice ind2
secuencia * n	la secuencia se repite n veces
secuencia1 + secuencia2	concatena las secuencias secuencia1 y secuencia2
objeto in secuencia	comprueba si objeto es un miembro de secuencia
objeto not in secuencia	comprueba si objeto no es un miembro de secuencia

La siguiente tabla muestra algunas de las funciones predefinidas que se pueden aplicar a las secuencias.

Tabla 10.2. Funciones Pre-Definidas

list (secuencia)	convierte la secuencia a un tipo lista
str (objeto)	convierte el objeto a un tipo string
tuple (secuencia)	convierte la secuencia a un tipo tupla
len (secuencia)	devuelve la longitud (numero de elementos) de la secuencia
max (secuencia)	devuelve el elemento mas grande de la secuencia
min (secuencia)	devuelve el elemento menor de la secuencia

Los diferentes tipos de secuencias se exponen a continuación.

10.6.1. Strings

Para Python, las cadenas (strings) son un conjunto contiguo de caracteres encerrados entre simples o dobles comillas.

```
>>> cadena='Hola Mundo!'
>>> print cadena
Hola Mundo!
```

Los strings son inmutables, no se puede alterar su valor a no ser que sean copiados a otro objeto string.

Los métodos y funciones aplicables al tipo de objeto string se encuentran definidos en el módulo string

```
>>> import string
>>> cadena.upper()
'HOLA MUNDO!'
```

La línea i**mport string** permite acceder a todos los métodos y atributos disponibles para un tipo de dato string, siendo la línea **cadena.upper()** la forma de invocar al método *upper* sobre el objeto string denominado *cadena*. Como resultado obtenemos el string en mayúsculas.

La siguiente tabla muestra algunos de los métodos disponibles en el módulo string:

Tabla 10.3. Métodos del módulo string

find(sub[, start[, end]])	devuelve el índice menor donde se encuentra el substring *sub* dentro del string
isalnum()	devuelve verdadero si todos los caracteres en el string son alfanuméricos y existe al menos uno.
isdigit()	devuelve verdadero si todos los caracteres en el string son dígitos y existe al menos uno.
lower()	devuelve una copia del string convertido a minúsculas.
split([sep [,maxsplit]])	devuelve una lista de elementos del string utilizando como separador *sep*.

10.6.2. Listas y Tuplas

Python posee varios tipos de datos para agrupar de una forma fácil diferentes valores. El tipo de dato más versátil es la lista. Una lista es una colección de elementos separados por comas y encerrados entre paréntesis. Los elementos de una lista no tienen porque ser del mismo tipo.

```
>>> lista = [ 1,'dos',3,'cuatro']
>>> print lista
[1, 'dos', 3, 'cuatro']
```

Como ocurría con el tipo de datos string, los índices de una lista empiezan en el 0, y pueden ser troceadas (sliced), concatenadas ...

```
>>> lista [0]
1
>>> lista [3]
'cuatro'
>>> lista [1:-1]
['dos', 3]
>>> lista + [5,'seis']
[1, 'dos', 3, 'cuatro', 5, 'seis']
```

A diferencia de los strings, que por definición eran inmutables, en una lista podemos cambiar el valor de un elemento individual.

```
>>> lista
[1, 'dos', 3, 'cuatro']
>>> lista [2] = 'tres'
>>> lista
[1, 'dos', 'tres', 'cuatro']
```

Es posible también crear listas anidadas (listas cuyos elementos pueden ser otras listas)

```
>>> lista1 = [2, 3]
>>> lista2 = [1, lista1, 4]
>>> len(lista2)
3
>>> lista2
[1, [2, 3], 4]
```

10.7. Diccionarios

El tipo de dato *diccionario* de Python es parecido al que se puede encontrar en otros lenguajes bajo el nombre de *Arrays Asociativos*. A diferencia de las secuencias donde los índices son números, los diccionarios están indexados por claves. Las claves solo podrán ser de algún tipo de dato inmutable (números y strings).

Un diccionario consiste por tanto en un conjunto de pares clave-valor, donde las claves son inmutables, mientras que los valores pueden ser de cualquier tipo.

Un diccionario se crea encerrando entre llaves ({}) una lista de pares clave-valor.

```
>>> usuario = {'login': 'fferrer', 'uid': 501 }
>>> usuario
{'login': 'fferrer', 'uid': 501}
>>> usuario ['login']
'fferrer'
>>> usuario ['uid']
501
```

Las operaciones habituales con diccionarios son la de almacenar y extraer algún valor con su correspondiente clave, pero también es posible eliminar algún elemento con la función **del()**.

A la hora de recorrer un diccionario será importante tener una lista de sus claves para poder acceder a los valores. El método **keys()** aplicado a un objeto de tipo diccionario, devolverá dicha lista.

```
>>> usuario
{'login': 'fferrer', 'uid': 501}
>>> usuario.keys()
['login', 'uid']
>>> del usuario ['uid']
>>> usuario
{'login': 'fferrer'}
```

En la siguiente tabla se exponen algunos de los principales métodos que se pueden usar con objetos del tipo diccionario.

Tabla 10.4. Métodos Pre-definidos de un diccionario

dict.clear()	elimina todos los elementos del diccionario *dict*
dict.get(clave,[default])	devuelve el valor de la clave o lo que definamos por defecto si la clave no se encuentra en el diccionario
dict.has_key(clave)	devuelve 1 si la clave se encuentra en el diccionario. En cualquier otro caso devuelve 0.
dict.items()	devuelve una lista de pares de tuplas clave-valor.
dict.keys()	devuelve una lista de claves
dict.values()	devuelve la lista de valores
dict.update(dict2)	añade los pares clave-valor del diccionario *dict2* al diccionario *dict*

10.8. Bloques de código

Una de las primeras cosas que sorprende cuando se empieza a utilizar Python es la forma en que este lenguaje delimita los bloques de código. No existen las llaves ({}) ni sentencias begin-end para encerrar el código, en su lugar un bloque de código viene delimitado por la identación.

```
>>> def f1 (a):
...     print a
...
>>> f1('Hola')
Hola
```

A parte de sorprender, lo que deja claro esta alternativa es que los programas en Python son legibles por cualquiera, lo cual a la larga es muy cómodo.

10.9. Sentencia if

La sintaxis de la sentencia if es la siguiente:

```
if expression1:
     if_bloque
elif expression2:
     elif_bloque
else:
     else_bloque
```

La expresión debe devolver un valor distinto de cero o verdadero para que se ejecute el if.

Otra cosa que también puede sorprender es que no existe una sentencia **case**, pero vista la sintaxis de la sentencia **if**, anidando diferentes elif podemos conseguir el mismo resultado.

10.10. Bucle while

La sintaxis y funcionamiento del bucle while es similar a la de la sentencia if:

```
while expression:
      while_bloque
```

El bloque while se ejecutará indefinidamente hasta que la expresión sea 0 o falso. De nuevo resaltar la identación para definir el bloque de código.

10.11. Bucle for

La sentencia **for** de Python es diferente a la de otros lenguajes. Solamente itera sobre una lista de elementos de una secuencia. En otros lenguajes (c o Perl) se puede iterar sobre una progresion aritmetica también.

La sintaxis es la siguiente:

```
for elemento in secuencia:
      bloque_for
```

Un ejemplo concreto de utilización:

```
>>> for i in [1,2,3,4]:
...     print i
...
1
2
3
4
```

si queremos tener un bucle **for** que itere sobre una progresión aritmética, podemos utilizar la función range() que devuelve una lista de números:

```
>>> for i in range(1,5):
...     print i
...
1
2
3
4
```

10.12. Definición de funciones

La palabra clave **def** es utilizada para la definición de una función. Debe de ir seguida del nombre de la función y la lista de parámetros entre paréntesis. Python no distingue entre procedimientos y funciones. Si es una función, esta devolverá algún tipo de valor con la sentencia **return**.

```
def nombre_funcion (param1, param2 ...):
     bloque_funcion
```

Los parámetros de una función pueden tener valores por defecto, de forma que cuando se invoque a la función no tengamos que especificarlos todos. En este último caso habrá que nominar los parámetros, para saber cuales toman un valor y cuales su defecto.

```
>>> def tabla_mult(p1=1):
...     for i in range(11):
...             print i * p1
...
```

10.13. Módulos

Los módulos son el mecanismo que utiliza Python para organizar trozos de código que luego puedan ser reutilizables. Los módulos pueden contener código ejecutable, clases o funciones.

Cuando se crea un fichero fuente de Python, el nombre del módulo será el nombre del fichero pero sin la extensión *.py*. Una vez que el módulo ha sido creado, la forma de utilizar sus componentes es invocando la orden **import** *nombre_módulo*. Y para hacer uso de las funciones o clases definidas en el módulo una vez que este ha sido importado, se utiliza la sintaxis típica de **modulo.funcion()**

A continuación se presenta el programa inicial Hola Mundo! pero utilizando las funciones de salida del módulo **sys**.

```
>>> import sys
>>> sys.stdout.write('Hello World!\n')
Hello World!
```

10.14. Ficheros

El soporte de acceso a ficheros es uno de los componentes más importantes de cualquier lenguaje. En Python existe una función pre-definida denominada **open** que permite abrir un fichero.

```
handle = open( file_name, access_mode='r')
```

La variable `file_name` contiene el nombre del fichero que deseamos abrir, mientras que el parámetro *acces_mode* define el modo en que queremos abrir el fichero. Estos modos pueden ser **r**(*lectura*), **w**(*escritura*), **a**(*añadir*), **b**(*binario*), **r+**(*lectura/escritura*)

Para leer los contenidos del fichero tenemos varios métodos. **readline()** leerá una línea en cada invocación del método, mientras que **readlines()** leerá todo el fichero generando una lista de líneas.

```
file = open(filename, 'r')
allLines = file.readlines()
file.close()
for eachLine in allLines:
   print eachLine,
```

A continuación se presenta una tabla con los métodos más utilizados sobre ficheros:

Tabla 10.5. Métodos del objeto Fichero

file.close()	cierra el fichero
file.fileno()	devuelve un entero representando el descriptor de fichero
file.flush()	descarga el buffer interno al fichero
file.read (size=-1)	lee todos los bytes del fichero o los especificados en el parámetro size
file.readline()	lee una línea del fichero incluyendo un salto de línea \n
file.readlines()	lee todas las líneas del fichero en un lista
file.seek(off, whence)	se mueve a una posición dentro del fichero off bytes desde lo que marque la variable whence (0= principio de fichero, 1=posición actual, 2=final de fichero)
file.write(str)	escribe la cadena str al fichero
file.writelines(list)	escribe la lista de cadenas al fichero

10.15. Errores y excepciones

Los errores de sintaxis se detectan en el proceso de compilación, pero Python puede detectar errores durante la ejecución del programa. Cuando se produce un error de ejecución, Python genera (*raises*) una excepción.

Para añadir este tipo de detección de errores, denominado manejo de excepciones, hay que encerrar nuestro código entre las clausulas **try-except**. El bloque que define la sentencia try será el código de nuestro programa en si. El código que viene detrás de la clausula except será el código que se ejecuta si se produjo alguna excepción. La sintaxis es la siguiente.

```
try:
     bloque_de_código
except Error:
     acción_contra_el_error
```

Vimos al principio de este capítulo como solicitar del usuario que introduzca un número y convertirlo a entero, ya que la entrada estandar siempre era un string. ¿ Pero que pasa si el usuario introduce una cadena ? Una pequeña modificación a nuestro ejemplo manejando la excepcion nos ayudará.

```
>>> while True:
...     try:
...         x = int(raw_input("Introduce un número: "))
...         break
...     except ValueError:
...         print "Oops! No es un número válido. Intentalo de nuevo..."
...
```


Si hubieramos ejecutado simplemente la sentencia que nos pide introducir el número y este no es correcto, esta hubiera fallado abortando el programa.

```
>>> x = int(raw_input("Introduce un número: "))
Introduce un número: aaaa
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): aaaa
```

Por tanto se ve claramente las posibilidades que tiene el manejo de excepciones para acotar firmemente nuestro programa.

11

Pluggable Authentication Modules (PAM)

Tabla de contenidos

11.1. Introducción

Los programas que permiten a los usuarios acceder a un sistema deben verificar la identidad del usuario a través de un proceso llamado autentificación. Históricamente, cada programa tiene su forma particular de realizar la autenticación. Bajo Fedora Linux, muchos de esos programas son configurados para usar un proceso de autenticación centralizado llamado Pluggable Authentication Modules (PAM).

PAM utiliza una arquitectura conectable y modular, que otorga al administrador del sistema una gran flexibilidad al establecer las políticas de autenticación para el sistema.

En la mayoría de los casos, el archivo de configuración por defecto PAM para una aplicación tipo PAM es suficiente. Sin embargo, algunas veces es necesario modificar el archivo de configuración. Debido a que un error en la configuración de PAM puede comprometer la seguridad del sistema, es importante comprender la estructura del estos archivos antes de hacer cualquier modificación.

PAM ofrece las ventajas siguientes:

- Un esquema de autenticación común que se puede usar con una gran variedad de aplicaciones.
- Permite gran flexibilidad y control de la autenticación para el administrador del sistema y el desarrollador de la aplicación.
- Los desarrolladores de aplicaciones no necesitan desarrollar su programa para usar un determinado esquema de autenticación. En su lugar, pueden concentrarse puramente en los detalles de su programa.

11.2. Ficheros de configuración

El directorio `/etc/pam.d/` contiene los archivos de configuración de PAM para cada aplicación tipo PAM. En versiones antiguas de PAM se utilizaba `/etc/pam.conf`, pero este archivo ya no se utiliza y `pam.conf` solamente es leído si el directorio `/etc/pam.d/` no existe.

Las aplicaciones tipo PAM o *servicios* tienen un archivo dentro del directorio `/etc/pam.d/`. Cada uno de estos archivos es llamado después del servicio para el cual controla el acceso.

Depende del programa tipo PAM definir el nombre de su servicio e instalar su archivo de configuración en el directorio /etc/pam.d/. Por ejemplo, el programa **login** define su nombre de servicio como `/etc/pam.d/login`.

11.3. Sintaxis de los ficheros de configuración

Cada archivo de configuración PAM contiene un grupo de directivas formateadas como sigue:

```
<module interface>  <control flag>   <module name>   <module arguments>
```

11.3.1. Interfaz de módulo

Existen cuatro tipos de módulos PAM usados para controlar el acceso a los servicios. Estos tipos establecen una correlación entre los diferentes aspectos del proceso de autorización:

- **auth** — Estos módulos autentifican a los usuarios pidiendo y controlando una contraseña. Los módulos con esta interfaz también pueden establecer credenciales, tales como pertenencias de grupo o tickets Kerberos.
- **account** — Estos módulos controlan que la autenticación sea permitida (que la cuenta no haya caducado, que el usuario tenga permiso de iniciar sesiones a esa hora del día, etc.).
- **password** — Estos módulos se usan para establecer y verificar contraseñas.
- **session** — Estos módulos configuran y administran sesiones de usuarios. Los módulos con esta interfaz también pueden realizar tareas adicionales que son necesitadas para permitir acceso, como el montaje de directorios principales de usuarios y hacer el buzón de correo disponible.

Un módulo individual puede proporcionar alguno o todas las interfaces de módulos mencionadas anteriormente. Por ejemplo, *pam_unix.so* tiene componentes que direccionan las cuatro interfaces.

En un archivo de configuración PAM, la interfaz del módulo es el primer aspecto a definir. Por ejemplo, una línea típica de una configuración sería:

```
auth      required  /lib/security/pam_unix.so
```

Esto provoca que PAM compruebe el componente *pam_unix.so* del módulo *auth*.

11.3.2. Apilar módulos

Las directivas de interfaces de módulos pueden ser definidas unas sobre otras (apiladas) para que se puedan usar varios módulos para un mismo propósito. El orden de una pila de módulos es muy importante en el proceso de autenticación.

El hecho de apilarlos hace que sea más fácil que el administrador exija diversas condiciones antes de permitir la autentificación del usuario. Por ejemplo, **rlogin** normalmente usa cinco módulos auth, como se puede ver en el archivo de configuración de PAM:

```
auth       required     /lib/security/pam_nologin.so
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_env.so
auth       sufficient   /lib/security/pam_rhosts_auth.so
auth       required     /lib/security/pam_stack.so service=system-auth
```

Antes de que a alguien se le permita llevar a cabo el rlogin, PAM verifica que el archivo `/etc/nologin` no exista, que no esté intentando iniciar una sesión en modo remoto como root y que se pueda cargar cualquier variable de entorno. Entonces se lleva a cabo una autenticación rhosts exitosa antes que se permita la conexión. Si falla la autenticación rhosts entonces se lleva a cabo una autenticación de contraseña estándar.

11.3.3. Indicadores de control

Todos los módulos PAM generan un resultado de éxito o fracaso cuando se les llama. Los indicadores de control le dicen a PAM qué hacer con el resultado. Como los módulos pueden apilarse en un determinado orden, los indicadores de control le dan la posibilidad de fijar la importancia de un módulo con respecto al objetivo final del proceso de autenticación para el servicio.

Hay cuatro indicadores de control definidos:

- **required** — El resultado del módulo debe ser exitoso para que la autenticación continue. Si un módulo required falla, el usuario no es notificado hasta que los resultados en todos los módulos referenciandos por esa interfaz sean completados.
- **requisite** — El resultado del módulo debe ser exitóso para que la autenticación continue. Sin embargo, si el resultado de un módulo requisite falla, el usuario es notificado inmediatamente con un mensaje reflejando el primer módulo required o requisite fracasado.
- **sufficient** — El resultado del módulo es ignorado si falla. Pero si el resultado del módulo con el indicador sufficient es exitoso y ningún módulo con indicador required ha fallado, entonces no se requiere ningún otro resultado y el usuario es autenticado para el servicio.
- **optional** — Se ignora el resultado del módulo si falla. Si el resultado del módulo es exitoso, no juega ningún papel en el éxito o fracaso general para el módulo. Un módulo con un indicador optional es necesario para la autenticación exitosa cuando no hay otros módulos referenciando la interfaz. En este caso, un módulo optional determina la autenticación PAM para esa interfaz.

Atención

El orden en el cual se llaman los módulos *required* no es crítico. Los indicadores de control *sufficient* y *requisite* provocan que el orden se vuelva importante.

11.3.4. Rutas de módulos

Las rutas de los módulos le indican a PAM dónde encontrar el módulo conectable que hay que usar con el tipo de interfaz de módulo especificada. Generalmente, se proporciona como una ruta completa al módulo, como `/lib/security/pam_stack.so`. Sin embargo, si no se proporciona la ruta entera, entonces se asume que el módulo está en el directorio `/lib/security/`, la dirección por defecto de los módulos PAM.

11.3.5. Argumentos de módulo

PAM utiliza argumentos para transmitir información a un módulo conectable durante la autenticación para algunos módulos.

Por ejemplo, el módulo `pam_userdb.so` usa contraseñas almacenads en una base de datos *Berkeley DB file* para autenticar a los usuarios. La base de datos Berkeley es una base de datos open source incorporada en muchas aplicaciones. El módulo toma un argumento db para que la base de datos Berkeley conozca que base de datos usar para el servicio solicitado.

Una línea típica `pam_userdb.so` dentro de un archivo PAM es similar a:

```
auth       required  pam_userdb.so db=<path-to-file>
```

En el ejemplo anterior, sustituiremos <path-to-file> con la ruta completa al archivo de base de datos Berkeley.

Los argumentos inválidos se ignoran y no afectan en ningún modo el éxito o fracaso del módulo PAM. Sin embargo, la mayoría de los módulos notificarán un error en el archivo `/var/log/messages`.

11.4. Ejemplos de configuración

A continuación una muestra de archivo de configuración de la aplicación PAM:

```
#%PAM-1.0
auth       required  /lib/security/pam_securetty.so
auth       required  /lib/security/pam_unix.so shadow nullok
auth       required  /lib/security/pam_nologin.so
account    required  /lib/security/pam_unix.so
password   required  /lib/security/pam_cracklib.so retry=3
password   required  /lib/security/pam_unix.so shadow nullok use_authtok
session    required  /lib/security/pam_unix.so
```

La primera línea es un comentario como lo es toda línea que inicie con el carácter (#).

Las líneas dos, tres y cuatro apilan tres módulos a usar para autenticaciones de inicio de sesión.

```
auth       required  /lib/security/pam_securetty.so
```

Este módulo se asegura de que si el usuario está tratando de conectarse como root, el tty en el cual el usuario se está conectando está listado en el archivo `/etc/securetty`, si ese archivo existe.

```
auth       required  /lib/security/pam_unix.so shadow nullok
```

Este módulo le solicita al usuario una contraseña y luego verifica la contraseña usando la información almacenada en `/etc/passwd` y, si existe `/etc/shadow`. El módulo `pam_unix.so` detecta automáti-

camente y utiliza contraseñas shadow para autenticar usuarios.

El argumento *nullok* instruye al módulo `pam_unix.so` a que permita una contraseña en blanco.

```
auth       required  /lib/security/pam_nologin.so
```

Este es el paso final de autenticación. Comprueba si el archivo `/etc/nologin` existe. Si nologin existe y el usuario no es root, la autenticación falla.

Nota

En este ejemplo, los tres módulos auth se ejecutarán, aún si el primer módulo auth falla. Esto previene al usuario de saber a qué nivel falla la autenticación. Tal conocimiento en las manos de una persona mal intencionada le permitiría violar el sistema fácilmente.

```
account    required  /lib/security/pam_unix.so
```

Este módulo realiza cualquier verificación de cuenta necesaria. Por ejemplo, las contraseñas shadow han sido activadas, el componente de la cuenta del módulo `pam_unix.so` verificará para ver si la cuenta ha expirado o si el usuario no ha cambiado la contraseña dentro del período de gracia otorgado.

```
password  required  /lib/security/pam_cracklib.so retry=3
```

Si la contraseña ha expirado, el componente de la contraseña del módulo pam_cracklib.so le pide una nueva contraseña. Luego evalúa la nueva contraseña para ver si puede ser fácilmente descubierta, ayudado por un programa que descubre las contraseñas basadas en diccionario. Si esto falla la primera vez, le dá al usuario dos oportunidades más de crear una contraseña más robusta debido al argumento retry=3.

```
password  required  /lib/security/pam_unix.so shadow nullok use_authtok
```

Esta línea especifica que si el programa cambia la contraseña del usuario, éste debería usar el componente password del módulo `pam_unix.so` para realizarlo. Esto sucederá tan sólo si la porción account del módulo pam_unix.so ha determinado que la contraseña necesita ser cambiada.

El argumento shadow le dice al módulo que cree contraseñas shadow cuando se actualiza la contraseña del usuario.

El argumento nullok indica al módulo que permita al usuario cambiar su contraseña desde una contraseña en blanco, de lo contrario una contraseña vacía o en blanco es tratada como un bloqueo de cuenta.

El argumento final de esta línea, use_authtok, proporciona un buen ejemplo de la importancia del orden al apilar módulos PAM. Este argumento advierte al módulo a no solicitar al usuario una nueva contraseña. En su lugar se acepta cualquier contraseña que fué registrada por un módulo de contraseña anterior. De este modo todas las nuevas contraseñas deben pasar el test de pam_cracklib.so para contraseñas seguras antes de ser aceptado.

```
session required /lib/security/pam_unix.so
```

La última línea especifica que el componente de la sesión del módulo `pam_unix.so` gestionará la sesión. Este módulo registra el nombre de usuario y el tipo de servicio en el fichero `/var/log/messages` al inicio y al final de cada sesión. Puede ser ampliado apilándolo con otros módulos de sesión si necesita más funcionalidad.

El próximo ejemplo de archivo de configuración ilustra el apilamiento del módulo auth para el programa rlogin.

```
#%PAM-1.0
auth       required    /lib/security/pam_nologin.so
auth       required    /lib/security/pam_securetty.so
auth       required    /lib/security/pam_env.so
auth       sufficient  /lib/security/pam_rhosts_auth.so
auth       required    /lib/security/pam_stack.so service=system-auth
```

Primero, `pam_nologin.so` verifica para ver si `/etc/nologin` existe. Si lo hace, nadie puede conectarse excepto root.

```
auth       required    /lib/security/pam_securetty.so
```

El módulo `pam_securetty.so` previene al usuario root de conectarse en terminales inseguros. Esto desactiva efectivamente a todos los intentos de root rlogin debido a las limitaciones de seguridad de la aplicación.

```
auth       required    /lib/security/pam_env.so
```

El módulo carga las variable de entorno especificadas en `/etc/security/pam_env.conf`.

```
auth       sufficient  /lib/security/pam_rhosts_auth.so
```

El módulo `pam_rhosts_auth.so` autentifica al usuario usando `.rhosts` en el directorio principal del usuario. Si tiene éxito, PAM considera inmediatamente la autenticación como exitosa. Si falla `pam_rhosts_auth.so` en autenticar al usuario, el intento de autenticación es ignorado.

```
auth       required    /lib/security/pam_stack.so service=system-auth
```

Si el módulo `pam_rhosts_auth.so` falla en autenticar al usuario, el módulo `pam_stack.so` realiza la autenticación de contraseñas normal.

El argumento service=system-auth indica que el usuario debe pasar a través de la configuración PAM para la autenticación del sistema como se encuentra en `/etc/pam.d/system-auth`.

11.5. Particularidades de Fedora Core 1

Fedora Core Linux permite al primer usuario que se conecte en una consola física de la máquina la habilidad de manipular algunos dispositivos y realizar algunas tareas normalmente reservadas para el usuario root. Esto es controlado por un módulo PAM llamado pam_console.so.

Cuando un usuario se registra en una máquina bajo Fedora Linux, el módulo `pam_console.so` es llamado por login o los programas de inicio de sesión gráfica, gdm y kdm. Si este usuario es el primero en conectarse en la consola física — llamado console user — el módulo concede la propiedad de una variedad de dispositivos que normalmente posee root. El usuario de la consola posee estos dispositivos hasta que la última sesión local para ese usuario finaliza. Una vez que el usuario se ha desconectado, la propiedad de los dispositivos vuelve a root.

Los dispositivos afectados incluyen, pero no son limitados, las tarjetas de sonido, las unidades de disco y las unidades de CD-ROM.

Esto permite que el usuario local manipule estos dispositivos sin llegar a ser root, de manera que se simplifican las tareas comunes para el usuario de la consola.

Modificando el archivo `/etc/security/console.perms`, el administrador puede editar la lista de dispositivos controlados por `pam_console.so`.

También se le permite al usuario de la consola el acceso a ciertos programas con un archivo que contenga el nombre del comando en el directorio `/etc/security/console.apps/`.

Un grupo notable de aplicaciones a las que tiene acceso el usuario de la consola son tres programas que cierran o abren el sistema. Estos son:

- **/sbin/halt**
- **/sbin/reboot**
- **/sbin/poweroff**

Debido a que estas son aplicaciones tipo PAM, ellas llaman al módulo `pam_console.so` como un requerimiento para el uso.

Para más información, consulte las páginas man para pam_console, console.perms, console.apps y userhelper.

A

Nota Legal

Administración Avanzada de Linux

por Fernando Ferrer García y Andrés Terrasa Barrena

Versión 1.0, Enero 2004