Servicios avanzados IV: Servidor Web Apache.

Autor: Enrique V. Bonet Esteban

Introducción.

El HTTP (HyperText Transfer Protocolo) es la base del armazón arquitectónico que se conoce actualmente como el World Wide Web[1], esto es la posibilidad de acceder a billones de documentos, distribuidos por millones de ordenadores en todo el mundo, documentos que a su vez se encuentran vinculados unos con otros.

Inicialmente la idea de la Web surgió en el laboratorio de altas energías del CERN, el Centro Europeo de Investigación Nuclear. La mayoría de los experimentos, altamente complejos y que requieren años de planteamiento y construcción de equipo, involucran a equipos multidisciplinares formados por personas de distintos países europeos. La Web surgió por la necesidad de lograr que estos equipos de investigadores, dispersos geográficamente por distintos países, tuvieran la posibilidad de colaborar de forma rápida y eficaz en el diseño y desarrollo de un conjunto rápidamente cambiante de informes, planos, dibujos, fotos y otros documentos.

La propuesta inicial de la Web de documentos vinculados surgió del físico del CERN Tim Berners-Lee en marzo de 1989. El primer prototipo (basado en texto) estaba en funcionamiento 18 meses después. En diciembre de 1991 se hizo una demostración pública en la conferencia Hypertext'91 en San Antonio (Texas, EE.UU.). El desarrollo continuó durante el siguiente año, culminando con la liberación de la primera interfaz gráfica, Mosaic, en febrero de 1993.

En 1994, el CERN y el M.I.T. firmaron un acuerdo para establecer el World Wide Web Consortium, una organización dedicada al desarrollo de la Web, la estandarización de protocolos y el fomento de interoperabilidad entre las instalaciones. Tim Berners-Lee se convirtió en el director.

Actualmente el Web es la herramienta más conocida y utilizada en la red Internet, siendo además la que más ha contribuido a popularizar la misma y fomentar su uso.

El servicio HTTP esta basado en una arquitectura cliente/servidor. En esta arquitectura la mayor complejidad, tanto de desarrollo como de administración, radica en el servidor, siendo el cliente tan solo[2] una herramienta capaz de ofrecer por pantalla los distintos elementos (documentos de texto, archivos de imágenes, etc.), que envía el servidor al cliente.

El estudio del tema lo desarrollaremos en tres puntos, en primer lugar un breve punto con la descripción del protocolo HTTP, un segundo punto con la explicación de la configuración del cliente y por último un tercer punto con la configuración del servidor.

[1] A partir de ahora lo denominaremos simplemente como Web.
[2] La afirmación debe ser entendida principalmente en el aspecto de administración, tal y como veremos a continuación.

El protocolo HTTP.

El protocolo estándar de transferencia de la Web es el HTTP (HyperText Transfer Protocol). Cada interacción consiste en una solicitud ASCII seguida de una respuesta de tipo MIME RFC 822[3]. Aunque la conexión de transporte se realiza mediante el protocolo TCP, el estándar no requiere formalmente su uso[4].

El protocolo HTTP consiste en dos elementos bastante diferentes: las solicitudes de los clientes a los servidores y las respuestas en el otro sentido. Aunque el HTTP se desarrollo inicialmente para usarse en la Web, ha sido generalizado con posterioridad en previsión de su utilización en futuras aplicaciones orientadas a objetos. Por esta razón, la primera palabra de la línea de solicitud completa es sencillamente el nombre del método (comando) a ejecutar y a continuación la página de la Web (u objeto general). Los métodos existentes se listan en la tabla siguiente, siendo sensibles al contexto (mayúsculas y minúsculas), por lo cual *GET* es un método valido pero *get* no lo es.

Método	**Descripción**
OPTIONS	Solicita información sobre las opciones de comunicación.
GET	Solicita recibir una página Web.
HEAD	Solicita leer la cabecera de una página Web.
POST	Añade información a un recurso nombrado.
PUT	Solicita almacenar una página Web.
DELETE	Elimina una página Web.
TRACE	Invoca la devolución del mensaje de solicitud.

El método *OPTIONS* solicita al servidor información sobre las opciones de comunicación disponibles para el recurso apuntado por un URL, generalmente un tipo MIME (text/html, etc.). De esta forma, el cliente puede determinar las posibilidades que tiene el servidor o las opciones asociadas a un recurso determinado.

El método *GET*[5] solicita al servidor que envíe la página codificada adecuadamente en MIME. Sin embargo, si a la solicitud *GET* le sigue una cabecera *If-Modified-Since*, el servidor sólo envía los datos si fueron modificados después de la fecha proporcionada. Usando este mecanismo, un navegador al que se solicitó una página que está en caché puede realizar una solicitud condicional al servidor.

El método *HEAD* simplemente pide la cabecera del documento, sin la página. Este método puede servir para obtener la hora de la última modificación, para recolectar información con fines de indexación, o simplemente para comprobar la validez de una URL transfiriendo menos datos que si se usa el método *GET*..

El método *POST* se utiliza para solicitar al servidor que acepte la información que se envía adjunta al mensaje. Este método se utiliza generalmente para la publicación de un mensaje en un grupo de noticias y para proporcionar un bloque de datos al servidor (por ejemplo los datos rellenados en un formulario por el usuario).

[3] El RFC 822 describe el formato estándar de intercambio de correo.
[4] Actualmente todos los servidores y clientes de HTTP utilizan el protocolo de transporte TCP, por lo que el desarrollo de un servidor o cliente que no utilice dicho protocolo de transporte no es aconsejable.
[5] Con posterioridad veremos más detalladamente el método GET.

El método *PUT* es el inverso de *GET*, en lugar de leer una página la escribe. Este método hace posible construir un conjunto de páginas de la Web en un servidor remoto. El cuerpo de la solicitud contiene la página y puede codificarse usando MIME, en cuyo caso las líneas que siguen a *PUT* deben incluir cabeceras *Content-Type* y de validación de identificación, para demostrar que el solicitante tiene permisos de ejecución de la operación.

El método *DELETE* elimina la página. Como con *PUT*, la validación de identificación y los permisos desempeñan un papel principal. No hay garantía de que *DELETE* tendrá éxito, puesto que, incluso si el servidor HTTP remoto está dispuesto a borrar la página, el archivo subyacente puede tener unos permisos de sistema que prohíban al servidor HTTP su modificación o eliminación.

Por último, el método *TRACE* se utiliza para depurar aplicaciones. El servidor final debe devolver el mensaje de solicitud, reflejando que ha recibido de forma correcta el mensaje o bien el tipo de error detectado.

Cada solicitud recibe una respuesta que consiste en una línea de estado y, posiblemente, información adicional (por ejemplo, toda o parte de una página Web). La línea de estado contiene un código que consiste en un número de tres dígitos y, posiblemente, un mensaje de texto aclaratorio del significado del código numérico. Un ejemplo de línea de estado es el siguiente:

```
HTTP/1.0 200 OK
```

Existen cinco tipos de códigos en función del primer dígito:

Código	**Descripción**
1xx	Informativo. No utilizado, reservado para usos futuros.
2xx	Éxito. La acción fue recibida y aceptada.
3xx	Redirección. Se necesita una acción adicional para llevar a cabo la solicitud.
4xx	Error del cliente. La solicitud contiene sintaxis errónea o no se puede conceder.
5xx	Error del servidor. El servidor no puede atender una solicitud aparentemente correcta.

El HTTP evoluciona constantemente. Se usan varias versiones y se están desarrollando otras. Las versiones se especifican mediante un sistema de numeración del tipo <*mayor*>.<*menor*> para indicar las versiones del protocolo. De esta forma el emisor puede indicar el formato del mensaje y su capacidad para entender futuras comunicaciones HTTP. La versión del mensaje HTTP se indica en el campo *HTTP-Version* en la primera línea del mensaje, como en el siguiente ejemplo:

```
HTTP-Version: HTTP/1.0
```

En caso de no especificarse la versión del protocolo, el receptor del mensaje asume que el mensaje tiene el formato *HTTP/1.0*.

Las dos versiones principales existentes actualmente son la *HTTP/1.0* y la *HTTP/1.1*. La diferencia principal entre ambas es que, mientras la versión 1.0 obliga a

que cada petición que un cliente realiza a un servidor genere una conexión TCP diferente, la versión 1.1 permite que una conexión albergue diferentes intercambios de solicitudes y respuestas.

El método GET.

De todos los métodos explicados con anterioridad, el método más usado es el método *GET*, que como hemos visto permite la solicitud de una página Web a un servidor por parte de un cliente. Las versiones actuales de HTTP reconocen dos tipos de solicitudes distintas del método *GET*: solicitudes sencillas y solicitudes completas.

Las solicitudes sencillas consisten en una única línea que comienza con el método *GET* y a continuación se encuentra el nombre de la página deseada, sin especificar la versión del protocolo y sin ningún dato adicional. Por tanto, su sintaxis es:

```
GET <página solicitada>
```

La respuesta que se obtiene del servidor no incluye ninguna línea con el estado, esto es, con el código de acierto o error de la solicitud enviada, consistiendo simplemente en una página sin ningún tipo de cabecera, sin ningún formato MIME y sin codificación alguna. Un ejemplo de solicitud sencilla es:

```
GET /home.html
```

Obteniendo como respuesta las siguientes líneas, que como puede observarse no van precedidas de ninguna línea de estado:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
...
</body>
</html>
```

Las solicitudes completas, las más usadas en la actualidad, se indican por la presencia de la versión del protocolo en la línea del método *GET*. A continuación viene una línea que indica el nombre del ordenador[6] al que se le realizó la petición de la página, y un conjunto de líneas subsiguientes que informan sobre la versión del cliente Web que envió la solicitud, los formatos MIME que son aceptados en la respuesta, etc., terminando la solicitud con una línea en blanco. Por tanto, la sintaxis de una solicitud completa es:

```
GET <página solicitada> <versión del protocolo HTTP>
Host: <nombre del servidor Web>
...
```

Dos ejemplos de solicitudes completas de páginas Web son los siguientes[7]:

[6] La utilidad de que en dicha línea figure el nombre del ordenador y no la dirección IP se entenderá cuando se vean con posterioridad los dominios virtuales en los servidores Web.

[7] El hecho de mostrar estos dos ejemplos concretos quedará claro cuando se explique, con posterioridad, la configuración del servidor Web del ordenador glup.irobot.uv.es.

```
GET /home.html HTTP/1.1
Host: irtic.uv.es
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.1)
Gecko/20021003
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,t
ext/plain;q=0.8,video/x-
mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive

GET /index.html HTTP/1.1
Host: www.cdlibre.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.1)
Gecko/20021003
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,t
ext/plain;q=0.8,video/x-
mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
```

La respuesta recibida contiene información sobre si la solicitud se proceso de forma correcta, la fecha y hora de la solicitud, fecha y hora de la última modificación de la página Web solicitada, tamaño de la página, formato MIME de la página enviada, etc. En nuestro caso, la respuesta a una de las solicitudes anteriores es la siguiente[8]:

```
HTTP/1.1 200 OK
Date: Sun, 17 Dec 2006 17:47:37 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Wed, 03 Nov 2004 16:42:35 GMT
ETag: "97864-2c0-3e7fd542b54c0"
Accept-Ranges: bytes
Content-Length: 704
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
```

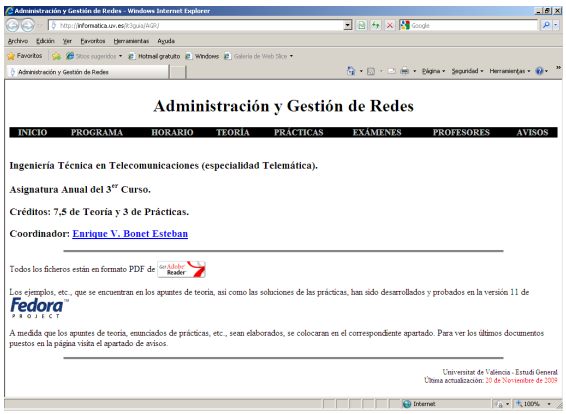
El cliente HTTP.

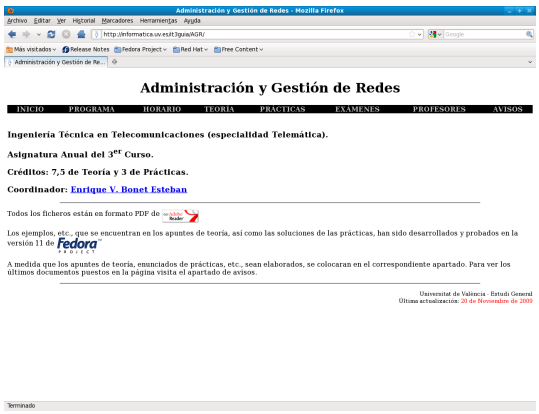
Desde el punto de vista del usuario, la Web consiste en un enorme conjunto a nivel mundial de documentos, llamados páginas. Cada página puede contener vínculos (enlaces) con otras páginas situadas en cualquier lugar del mundo. Los usuarios pueden

[8] Seguidas dichas líneas de cabecera, obviamente, del contenido de la página HTML solicitada, que aquí no es mostrada.

seguir un vínculo (por ejemplo, haciendo clic en él), lo que los lleva a la página vinculada. Este proceso puede repetirse indefinidamente.

Las páginas Web se ven mediante un programa llamado navegador[9]. El navegador solicita una página Web, espera la recepción de la página solicitada y, una vez obtenida, interpreta el texto recibido[10] y los comandos para dar formato al texto que contiene la página y la muestra en la pantalla. Un ejemplo de página Web vista en los dos navegadores más utilizados puede verse a continuación:





En la actualidad, existen páginas que contienen pistas de audio, fragmentos de vídeo, etc[11]. En tal caso los navegadores revisan un archivo de configuración para ver el modo de mostrar dichos datos al usuario. Generalmente el archivo de configuración indica el nombre del programa, llamado visor externo o aplicación ayudante, que se ejecutará con la página Web traída como entrada. Si no existe un visor para ese tipo de datos el navegador solicita al usuario que escoja uno.

La configuración de un cliente Web es relativamente sencilla, bastando con instalar de forma correcta el navegador, así como todos los visores externos que queramos utilizar, para que este funcione. La única dificultad que puede presentarse es la necesidad de configurar el servidor proxy[12] de la red, de forma que se nos permita el acceso a las páginas Web externas a nuestra subred.

Dicha configuración es sencilla, así en el servidor Web Microsoft Explorer la configuración se realiza mediante el acceso a los siguientes menús y submenús: Herramientas Opciones de Internet Conexiones Configuración de LAN. Una vez aquí, marcamos la casilla “Utilizar un servidor proxy para su LAN...” , y si deseamos también la casilla “No utilizar servidor proxy para direcciones locales”, pulsamos en Opciones Avanzadas y en la línea de HTTP ponemos como servidor proxy el servidor proxy de nuestra red y como puerto el que utilice nuestro servidor proxy.

[9] Actualmente los navegadores más utilizados son Internet Explorer, Mozilla Firefox y Google Chrome.
[10] Inicialmente las páginas Web estaban escritas mediante HTML. En la actualidad las páginas Web han evolucionado mezclado otros lenguajes como Javascript, etc.
[11] El resultado de mezclar páginas de hipertexto con otros medios se conoce con el nombre de hipermedia.
[12] Un servidor proxy es un programa que gestiona las conexiones Web de una red, almacenando las páginas recibidas, de forma que posteriores peticiones de las mismas páginas Web no tengan que ser solicitadas al servidor que las contiene, sino que el programa proxy proporcione dichas páginas Web, mejorando la velocidad de respuesta y disminuyendo la congestión en la red.

De igual forma, en el cliente Web Firefox, distribuido generalmente con el sistema operativo Linux, la configuración se realiza mediante el acceso a los menús y submenús: Editar Preferencias Avanzado Red Configuración. Una vez aquí marcamos la casilla "Configuración Manual del Proxy" y escribimos en la línea de Proxy HTTP el servidor proxy y su puerto. Además, podemos poner en la línea de "No Proxy para:" la red uv.es, pues corresponde a nuestra red local.[13]

La solicitud de una página Web determinada se realiza mediante la introducción en el navegador Web de lo que se conoce como "Localizador Universal de Recursos", URL (Universal Resource Locator). El URL de una página Web esta formado por tres campos, de acuerdo a la siguiente sintaxis:

```
http://<nombre del ordenador>[:puerto][/<página Web solicitada>]
```

Donde *"nombre del ordenador"* es el nombre del ordenador donde se encuentra la página Web que deseamos visualizar. *"puerto"* es un parámetro optativo que indica el número de puerto TCP que utiliza el servidor de HTTP[14] y *"página Web solicitada"* es el nombre de la página Web solicitada. Este parámetro es optativo, aunque suele aparecer casi siempre, pues en ciertas configuraciones del servidor, si no es solicitada ninguna página Web, el servidor asume por defecto que se solicita una página Web de una lista de páginas Web por defecto que tiene configurada, devolviendo la primera de las páginas Web por defecto que se encuentren según el orden preestablecido.

El servidor HTTP.

El servidor HTTP es el programa que atiende las peticiones de los clientes Web y proporciona las páginas solicitadas. El servidor HTTP utiliza, de forma general, el puerto 80 TCP para la recepción de las peticiones de los clientes[15]. Aunque existen un gran número de servidores Web disponibles, nosotros nos centraremos en el servidor Web conocido con el nombre de Apache, que es el que se encuentra disponible en el sistema operativo Linux[16].

El servidor Web Apache se encuentra en */usr/sbin/httpd*. Las opciones más frecuentes son *-d <directorio>* y *-f <fichero>*.

La opción *-d <directorio>* indica el directorio raíz donde se encuentran las páginas Web del servidor, dicha opción puede ser sobrescrita por la entrada que especifica el directorio raíz en el fichero de configuración[17].

La opción *-f <fichero>* indica el fichero de configuración que debe utilizarse en lugar del fichero de configuración por defecto. Si la entrada comienza por "/" se supone

[13] En ambos casos hemos supuesto que deseamos hacer la configuración de forma manual. Existen otras opciones que permiten detectar de forma automática el servidor proxy de la red, y para las cuales basta con activar la casilla adecuada.
[14] El puerto 80 TCP es el que tiene asignado por defecto el servicio de Web. Sin embargo, un servidor Web puede instalarse en cualquier puerto TCP que se encuentre disponible.
[15] El puerto 443 TCP es usado para las peticiones de conexiones seguras, esto es, conexiones realizadas mediante el uso de SSL.
[16] Además de para el sistema operativo Linux, existen servidores Web Apache para otros sistemas operativos como pueden ser Windows, Solaris e IRIX.
[17] La estructura del fichero de configuración la veremos con posterioridad.

un camino absoluto, en caso contrario se supone un camino añadido al directorio indicado por la opción anterior, si dicha opción se encuentra presente, o bien al directorio por defecto en caso contrario.

Merece la pena comentar la existencia de la opción *-t* que indica al servidor que compruebe si el fichero de configuración es correcto o no. Esta opción en ningún caso ejecuta el servidor, tan solo comprueba la validez del fichero de configuración y muestra el resultado, terminando su ejecución. Si a la opción *-t* se le añade la opción *-D DUMP_VHOSTS* se muestran detalles de los servidores virtuales que estén configurados, mientras que si se añade la opción *-D DUMP_MODULES* se muestran detalles de los módulos que el servidor ejecuta.

Los ficheros de configuración por defecto del servidor Web Apache se encuentra dentro del directorio */etc/httpd*. En dicho directorio se pueden observar la presencia de tres subdirectorios, *conf*, *conf.d* y *conf.modules.d*. En el primero de dichos subdirectorios es donde se encuentra el fichero que configura propiamente el servidor Web, mientras que en los otros dos directorios se encuentran los ficheros que configuran las extensiones del servidor Web, esto es, aquellos módulos que permiten al servidor Web establecer conexiones seguras mediante SSL (ficheros *conf.d/ssl.conf* y *conf.modules.d/00-ssl.conf*), consultar bases de datos y crear páginas Web dinámicas en función de las respuestas obtenidas (fichero *conf.d/php.conf* y *conf.modules.d/10-php.conf*), etc. Esta estructura es muy práctica, pues al añadir o quitar una extensión basta con insertar o borrar sus ficheros de configuración del directorio de extensiones, sin que sea necesario modificar para nada el fichero de configuración del servidor propiamente dicho.

La configuración de las extensiones del servidor Web es sencilla, por ejemplo, el fichero de configuración de la extensión para que el servidor pueda atender peticiones de páginas de php (ficheros *conf.modules.d/10-php.conf* y *conf.d/php.conf*) es:

```
# Fichero conf.modules.d/10-php.conf
<IfModule prefork.c>
  LoadModule php5_module modules/libphp5.so
</IfModule>
<IfModule !prefork.c>
  LoadModule php5_module modules/libphp5-zts.so
</IfModule>

# Fichero conf.d/php.conf
<FilesMatch \.php$>
   SetHandler application/x-httpd-php
</FilesMatch>
AddType text/html .php
DirectoryIndex index.php
php_value session.save_handler  "files"
php_value session.save_path     "/var/lib/php/session"
```

En el primer fichero las líneas indican el nombre de registro del modulo que debe cargar el servidor para poder atender peticiones de PHP y donde se encuentra, estas líneas indican que se cargue un módulo u otro según el modo de funcionamiento del servidor Apache. En el segundo fichero las primeras cuatro líneas indican las extensiones y el tipo de respuesta de las peticiones que serán atendidos por este modulo

(extensión PHP que corresponde a ficheros de tipo texto HTML). La siguiente línea añade una entrada nueva (*index.php*) a las entradas por defecto de los ficheros a servir por defecto si no se especifica ninguno en la solicitud. Por último, las dos últimas líneas ajustan valores de configuración de la extensión PHP para que funcione de forma correcta con el servidor web Apache.

Apache tiene dos modos de funcionamiento diferente, eligiendo el modo de funcionamiento (modo *prefork* o modo *worker*) en la compilación del servidor. En modo *prefork* el servidor funciona de forma que un proceso padre ejecuta procesos hijo para que cada uno atienda las peticiones de los clientes, mientras que en el modo *worker* un proceso padre ejecuta procesos hijo que lanzan hilos para atender las peticiones de los clientes. El modo en que se encuentra compilado el servidor puede obtenerse ejecutando el comando *httpd -V* y buscado en la salida la línea *"Server MPM:"* o bien ejecutando directamente *httpd -V|grep "Server MPM:"* aunque de forma general en UNIX/Linux es *prefork*[18].

El fichero de configuración del servidor Web Apache, que como hemos comentado se encuentra en dentro del subdirectorio *conf* tiene como nombre *httpd.conf*, estando dividido en tres secciones: Configuración global, configuración general del servidor y configuración de los servidores virtuales. En todas las secciones las líneas que comienzan por el carácter # son consideradas como comentarios.

Configuración global.

La sección de configuración global es la que se encarga de especificar el funcionamiento del servidor Web, indicando el directorio de los ficheros de configuración, el modo de funcionamiento del servidor, etc. Las entradas básicas de la configuración global son[19]:

```
ServerRoot "/etc/httpd"
```

Indica el directorio raíz del servidor bajo el que se encuentran los ficheros de configuración, error y los de log, en este caso *"/etc/httpd"*. Es importante comentar que no puede añadirse el carácter "/" al final del mismo.

```
Listen 147.156.222.65:80
```

Continuando con la configuración, la directiva *Listen* especifica la dirección IP y puerto en que el servidor escucha las conexiones. La especificación de la dirección IP es opcional y si no se indica, o bien es un asterisco, el servidor escuchará las peticiones de todos los interfaces de red existentes en el servidor. Pueden utilizarse múltiples directivas *Listen* para especificar distintos interfaces de red y/o puertos. Por ejemplo:

```
Listen *:80
Listen 127.0.0.1:8080
```

[18] En el apéndice A se puede encontrar una breve explicación de las opciones de configuración de ambos modos, que en la configuración actual del servidor utilizan los valores por defecto, validos en la mayoría de usos del servidor.

[19] Una explicación más detallada de todo lo comentado en estos apuntes puede encontrarse en la URL: http://httpd.apache.org/docs/2.4.

Indica que se escuche en el puerto 80 en todos los interfaces de red y en el puerto 8080 por el interfaz de loopback. De igual forma:

```
Listen 147.156.223.157:80
Listen [2001:720:1014:222::2]:80
```

Indica que escuche en el puerto 80 en la dirección IPv4 147.156.223.157 y en la dirección IPv6 2001:720:1024:222::2.

```
Include conf.modules.d/*.conf
```

Indica que incluya todos los ficheros de extensión **.conf* que se encuentran dentro de *conf.modules.d* y que contienen las librerías dinámicas necesarias para el funcionamiento de las extensiones del servidor web.

```
User apache
Group apache
```

Por último, estas líneas especifican como que usuario (*User*) y como que grupo (*Group*) se ejecutará el servidor Web. Esta es una opción de seguridad y permite restringir el acceso del servidor a directorios, tanto en un funcionamiento correcto como ante un fallo en el mismo.

Configuración general del servidor.

La configuración general del servidor es el que define el comportamiento del servidor por defecto y de todos los servidores virtuales que se puedan definir, excepto que en los mismos se especifique otra opción.

```
ServerAdmin webmaster@irtic.uv.es
```

Indica la dirección de correo del administrador del servidor.

```
ServerName irtic.uv.es
```

La opción *ServerName* especifica el nombre y puerto con el que el servidor se identificará ante las peticiones que se realicen. Si no se indica el valor del nombre este se obtiene realizando una consulta inversa de DNS (dirección IP a nombre)[20], y si no se especifica el puerto se utilizará el puerto por el que se recibió la petición[21].

```
<Directory />
    AllowOverride None
    Require all granted
</Directory>
```

Especifica opciones de control de acceso a las páginas web.

```
DocumentRoot "/var/www/html"
```

[20] Si un ordenador posee un nombre y varios alias en el DNS el valor que devolverán el DNS será el del nombre y será este el utilizado.

[21] Téngase en cuenta que pueden especificarse más de un puerto de escucha con múltiples líneas *Listen* en la configuración.

Indica el directorio a partir del cual se encuentran las páginas Web del servidor.

```
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>

<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Especifican opciones de control de acceso a las páginas web. Estás opciones serán explicadas con posterioridad en el control de acceso a las páginas web.

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

Indica los nombres y el orden de las páginas por defecto que el servidor Web buscará si en la petición de un cliente no se especifica un nombre de página. Esto permite que solicitudes como *http://www.google.com*, etc., puedan ser atendidas sin que se especifique la página Web solicitada, pues como podemos ver en esa petición solo se especifica el nombre del servidor, sin indicar el nombre de la página que deseamos que sea servida.

```
<Files ".ht*">
    Require all denied
</FilesMatch>
```

Especifica las reglas para evitar que los ficheros que controlan el acceso a determinadas páginas web puedan ser accedido por un cliente web. Se analizarán con más detalle posteriormente en el apartado dedicado al control de acceso a las páginas web[22].

```
ErrorLog "logs/error_log"
```

Indica donde se escribirán los mensajes de error[23].

```
LogLevel warn
```

Indica el nivel de detalle de los mensajes de error que se almacenan. Los niveles existentes son *emerg*, *alert*, *crit*, *error*, *warn*, *notice*, *info* y *debug*, de mayor a menor importancia. La especificación de un nivel implica que todos los mensajes de niveles superiores también serán escritos en el log de error del sistema.

```
<IfModule log_config_module>
```

[22] Esta entrada esta pensada si el usuario utiliza los nombres de ficheros estandar, que son *.htaccess* y *.htpasswd*.

[23] Generalmente los mensajes de error se producen cuando se solicita una página que no existe en el servidor o bien, que este ha sido incapaz de servir por cualquier motivo.

```
   LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
{User-Agent}i\"" combined
   LogFormat "%h %l %u %t \"%r\" %>s %b" common
   <IfModule logio_module>
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
{User-Agent}i\" %I %O" combinedio
   </IfModule>
   CustomLog logs/access_log common
<IfModule>
```

Las entradas *LogFormat* especifican distintos tipos de información que debe escribirse en el fichero de logs de acceso del sistema, así como un nombre para las referirse a las mismas[24], mientras que *CustomLog* indica el fichero donde se escribirán los mensajes con la información de las direcciones IP, páginas solicitadas, etc., por los clientes del servidor, y la información que se almacenará en función del nombre especificado.

```
<IfModule alias_module>
   ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
</IfModule>

<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None
    Require all granted
</Directory>
```

Una entrada *Alias* en general, y *ScriptAlias* en particular, permite especificar un camino distinto al camino por defecto para un recurso al que debe acceder el servidor. En este caso se indica que si se solicita un script CGI con la URL *http://servidor/cgi-bin/nombre* se acceda realmente al directorio */var/www/cgi-bin* para encontrar el script con el nombre especificado y no al directorio "valor de *DocumentRoot"/cgi-bin* como debería realizarse de forma normal anteponiendo el camino por defecto indicado en *DocumentRoot*. Por su parte el resto de líneas indican las condiciones de acceso a dichos ficheros.

```
<IfModule mime_module>
   TypesConfig /etc/mime.types
   AddType application/x-compress .Z
   AddType application/x-gzip .gz .tgz
   AddType text/html .shtml
   AddOutputFilter INCLUDES .shtml
</IfModule>
```

Define los tipos de ficheros que conoce el servidor web y añade la definición de algunos tipos de ficheros en función de su extensión. Además, se indica que los ficheros de extensión *.shtml* deben ser incluidos en el fichero que los incluye y no enviados de forma individual.

```
AddDefaultCharset UTF-8
```

[24] Una descripción de la información que puede escribirse en el fichero de log de acceso se encuentra en la URL *http://httpd.apache.org/docs/2.4/es/mod/mod_log_config.html#formats*.

Indica el conjunto de caracteres por defecto a utilizar.

```
<IfModule mime_magic_module>
   MIMEMagicFile conf/magic
</IfModule>
```

Indica el fichero que contiene la especificación de tipos MIME de los ficheros en función de los códigos MIME.

```
EnableSendFile on
```

Indica que el servidor web Apache utilice el soporte de enviar ficheros del kernel sin necesidad de que el servidor los lea. Esto es útil cuando el contenido del fichero es estático y por tanto Apache no debe modificar el mismo.

Configuración de los servidores virtuales.

Antes de explicar la configuración de los servidores virtuales conviene aclarar en que consisten y cual es su utilidad.

Un servidor virtual es una entrada en un servidor Web que atiende las peticiones realizadas a una URL que no se corresponde con el nombre del servidor principal, y que sin embargo ha sido enviada al ordenador donde se encuentra el servidor Web por ir dirigida a la misma dirección IP.

El motivo de su aparición y utilidad es la posibilidad de que un solo ordenador de alojamiento a múltiples dominios de páginas Web, de forma que una sola dirección IP, la del ordenador, pueda responder a las peticiones de páginas Web correspondientes a distintos nombres de dominio.

Es necesario resaltar en este punto, que la configuración de los servidores virtuales desactiva automáticamente el servidor principal, por lo que el mismo debe incluirse como un servidor virtual más.

En nuestro ejemplo, el servidor Web instalado, además de dar servicio al dominio irtic.*uv.es* como servidor principal, da servicio al dominio *www.cdlibre.org*, dominio virtual que se encuentra asignado a la dirección IP de nuestro servidor[25].

Las entradas más comunes en la configuración de los servidores virtuales son:

```
<VirtualHost 147.156.222.65:80>
      ServerName irtic.uv.es
      DocumentRoot /var/www/html
      ServerAdmin webmaster@irtic.uv.es
      ServerSignature email
      DirectoryIndex home.html index.html
</VirtualHost>

<VirtualHost 147.156.222.65:80>
```

[25] La configuración de servidores virtuales elimina el uso del servidor por defecto, por lo que si se desea continuar usando el mismo debe incluirse como un servidor virtual más.

```
      ServerName www.cdlibre.org
      ServerAlias cdlibre.org *.cdlibre.org
      DocumentRoot /home/barto/public_html
      ServerAdmin barto@glup.uv.es
      ServerSignature email
      DirectoryIndex home.html index.html
      ErrorLog /home/barto/error.log
      TransferLog /home/barto/transfer.log
</VirtualHost>
```

Las entradas *<VirtualHost nombre[:puerto]>* y *</VirtualHost>* permiten indicar, mediante el nombre o dirección IP, la dirección y puerto en que escucha el servidor virtual al que se refieren las entradas comprendidas entre ellas, permitiendo limitar la especificación de la entrada *NameVirtualHost* para este servidor virtual en particular.

Las entradas *ServerName* indican el nombre del servidor que debe solicitar el cliente para que corresponda con el aquí indicado y, por tanto, sea atendido por este servidor Web virtual.

Por su parte, la entrada *ServerAlias* indica otros nombres que puede tener este mismo servidor virtual y que deben ser atendidos por el mismo. Así, el servidor de nombre *www.cdlibre.org* atenderá las peticiones a *cdlibre.org* o a cualquier otro dominio que termine con *cdlibre.org* (entrada **.cdlibre.org*).

Las entradas *DocumentRoot*, *ServerAdmin*, *ServerSignature* y *DirectoryIndex* tienen idéntico significado que sus homologas del servidor principal.

Por último, las entradas *ErrorLog* y *TransferLog* indican la localización de los ficheros donde queremos que vayan los mensajes de error y de páginas solicitadas y transferidas para este servidor virtual.

Antes de terminar es necesario resaltar dos aspectos de los servidores virtuales. El primero es que cualquier valor no especificado en la definición del servidor virtual tomara el valor que tiene en la definición del servidor por defecto. Así, por ejemplo, como en el servidor virtual *irtic.uv.es* no hemos especificado las entradas *ErrorLog* y *TransferLog*, los mensajes de error y de las páginas transferidas a los clientes son almacenados en los ficheros correspondientes del servidor principal.

El segundo es que en caso de que una URL haga referencia a un ordenador con servidores virtuales, y esa URL no se corresponda con el nombre de ningún servidor virtual, el servidor virtual que atiende la petición es el primero de todos los que aparecen. De este modo, si borramos la entrada del servidor virtual correspondiente a *www.cdlibre.org*, toda petición que se realizará a ese servidor virtual mostraría las páginas Web del servidor irtic.*uv.es*.

Control de acceso a los recursos.

El control de acceso a los recursos puede realizarse de dos formas distintas no excluyentes, el control de acceso por dirección IP del cliente y el control de acceso por usuario.

Control de acceso por dirección IP del cliente.

El acceso a los recursos por dirección IP del cliente es controlado por Apache mediante la declaración de distintas entradas que se aplican a un directorio del sistema de ficheros, una URL, etc. La especificación de las entradas y sus condiciones de control de acceso puede realizarse de tres formas:

```
<Directory "directorio">
...
</Directory>

<Location "URL">
...
</Location>

<Files "fichero">
...
</Files>
```

La entrada *Directory* implica que las órdenes especificadas en el control de acceso se aplican al directorio especificado y sus subdirectorios, excepto que exista una regla más específica para alguno de los subdirectorios. Por ejemplo, la entrada:

```
<Directory "/">
...
</Directory>
```

Indica que las condiciones de control de acceso se aplican a todos los directorios del ordenador, mientras que la entrada:

```
<Directory "/var/www/html">
...
</Directory>
```

Indica que las condiciones se apliquen al directorio */var/www/html* y sus subdirectorios.

Por otra parte, la entrada *Location* implica que las órdenes de control de acceso se aplican a la URL especificada, por lo que su camino es relativo respecto al valor de la raíz de los documentos del servidor[26]. Por ejemplo, la entrada:

```
<Location "/">
...
</Location>
```

Indica que las condiciones de control de acceso se apliquen a todos los subdirectorios dentro de */var/www/htm*, si el valor de *DocumentRoot* es */var/www/html*, y la entrada:

```
<Location "/monitor">
```

[26] El directorio raíz de los documentos del servidor viene dado por la variable de configuración *DocumentRoot* tal y como vimos.

```
...
</Location>
```

Indica que las condiciones de control de acceso se apliquen a todos los subdirectorios dentro de */var/www/html/monitor*.

Es importante resaltar en este punto que se ha de tener cuidado con el uso de *Location* para limitar el acceso a elementos, pues se aplica a la URL especificada y no al elemento, pudiendo suceder que si dos URLs hacen referencia al mismo elemento, una pueda estar limitada por el uso de *Location* mientras que la otra no, pudiendo entonces acceder al elemento y saltarse las reglas de control de acceso mediante esta última referencia.

Por último, la entrada *Files* se refiere al nombre de archivo especificado, independientemente del directorio donde se encuentre el archivo. Así, la entrada:

```
<Files "privado.html">
...
</Files>
```

Se aplicará a todos los ficheros de nombre *privado.html*, independientemente del directorio donde se encuentren el fichero.

Todas las entradas anteriores tienen sus equivalentes *DirectoryMatch*, *LocationMatch* y *FilesMatch*, que permiten utilizar expresiones regulares en el nombre, de forma que se pueda hacer referencia a múltiples directorios o ficheros. Así:

```
<FilesMatch "^\.ht">
...
</FilesMatch>
```

Indica cualquier fichero que comience por *.ht*[27].

El orden en que se aplican las entradas puede tener importancia en su funcionamiento, por ello es necesario tener en cuenta que en primer lugar se aplican las secciones *Directory* existentes[28], luego las secciones *DirectoryMatch*, a continuación *Files* y *FilesMatch* sin ninguna prioridad entre ellas, y por último *Location* y *LocationMatch*, también sin ninguna prioridad entre ellas. Por ello, si analizamos el siguiente ejemplo:

```
<Location "/">
D
</Location>

<Files "fichero.html">
C
</Files>
```

[27] Aunque también es posible utilizar expresiones regulares en las entradas <Directory>, <Location> y <Files>, precediendo las mismas del símbolo ~, es recomendable utilizar sus equivalentes para expresiones regulares <DirectoryMatch>, <LocationMatch> y <FilesMatch>.

[28] Si se habilita el uso de control de acceso de usuario a los directorios, que veremos a continuación, estas restricciones de control tienen siempre prioridad sobre cualquier sección, incluida la sección *Directory*.

```
<DirectoryMatch "^.*b$">
B
</DirectoryMatch>

<Directory "/">
A
</Directory>
```

El orden de evaluación de las mismas será A, B, C y por último D.

Dentro de las entradas especificadas de esta forma se indican las directivas que permiten el acceso o la denegación de acceso a cada una de ellas. La directiva que controla este acceso es *Require*.

La directiva *Require* usada para controlar el acceso por dirección IP puede tomar las formas:

```
Require all granted
Require all denied
Require [not] host {nombre|dominio}
Require [not] ip {ip|subred}
```

Require all granted y *Require all denied* indican, respectivamente, que todos los ordenadores tienen permitido el acceso o denegado el acceso y son las únicas directivas que permiten o deniegan el acceso. Por su parte *Require host {nombre|dominio}* indica que si el ordenador o dominio se cumple el requisito es verdad (o no es verdad si esta precedida de *not*). Por último, *Require ip {ip|subred}* indica que si la dirección IP o la red/subred se cumple el requisito es verdad (o no es verdad si esta precedida de *not*)[29]. Así, por ejemplo:

```
Require ip 147.156.0.0/16
```

Será verdad si el ordenador es de la UV, mientras que:

```
Require not host irobot.uv.es
```

No será verdad si el ordenador es del IRTIC y será verdad en caso contrario.

La directiva *Require* puede combinarse mediante las directivas *<RequireAll>...</RequireAll>*, *<RequireAny>...</RequireAny>* y *<RequireNone>...</RequireNone>* para formar reglas más complejas. *<RequireAll>...</RequireAll>* indica que la autenticación se cumple si ninguna de las directivas que se encuentran en su interior fallan y al menos una de ellas se cumple. Por su parte, <RequireAny>...</RequireAny> indica que la autenticación se cumple si al menos una de las directivas se cumple. Por último, <RequireNone>...</RequireNone> indica que la autenticación se cumple si todas las directivas no se cumplen. Por ejemplo:

```
<RequireAll>
   Require all granted
```

[29] Es necesario resaltar que *not* no implica una negación lógica y por tanto un valor false, sino que únicamente indica que no sea verdad lo indicado.

```
   Require not ip 147.156.0.0/16
</RequireAll>
```

Permitirá el acceso si el ordenador no es de la Universidad de Valencia. Otro ejemplo es el siguiente:

```
<RequireAny>
   Require all denied
   Require ip 147.156.222.0/23
   Require host www.uv.es
</RequireAny>
```

Que denegará el acceso si la dirección IP es de la subred del IRTIC o el ordenador es www.*uv*.*es*. Un último ejemplo es el siguiente:

```
<RequireNone>
   Require all granted
   Require not ip 147.156.0.0/16
</RequireNone>
```

Que permitirá el acceso si el ordenador es de la Universidad de Valencia.

Un ejemplo en el cual se muestra la importancia del orden en que se evalúan las entradas es el siguiente[30]:

```
<Location "/">
    Require all granted
</Location>

<Directory "/var/www/html">
   Require all denied
</Directory>
```

En el que podemos ver como la entrada *Directory* restringe el acceso al directorio raíz de las páginas Web, pero la entrada *Location* permite el acceso a dicho directorio raíz. Al evaluarse en último lugar la entrada *Location*, la restricción de la entrada *Directory* queda sin efecto.

Además de las condiciones de control de acceso, dentro de las entradas es posible especificar algunas funcionalidades adicionales que estarán disponibles en el directorio especificados. Estas funcionalidades adicionales se indican, dentro de la entrada, como:

```
Options <funcionalidad> [... funcionalidad]
```

Los posibles valores de *Options* se encuentran en la siguiente tabla:

Valor	**Descripción**
`None`	Ninguna funcionalidad adicional está activa
`All`	Todas las funcionalidades adicionales menos *MultiViews*.
`ExecCGI`	Permite utilizar scripts CGI.

[30] En el ejemplo suponemos que el directorio raíz de las páginas web es /var/www/html.

Valor	Descripción
`FollowSymLinks`	Permite seguir los enlaces simbólicos[31].
`SymLinksIfOwnerMatch`	Permite seguir los enlaces simbólicos solo si el fichero o directorio final tiene el mismo dueño que el enlace[32].
`Includes`	Permite incluir SSI[33].
`IncludesNoExec`	Permite incluir SSI pero excluyendo aquellos que ejecutan comandos o CGIs.
`Indexes`	Muestra un listado con el contenido del directorio si no existen los archivos especificados en *DirectoryIndex*.
`MultiViews`	Permite la negociación del contenido, por ejemplo, el uso de distintos lenguajes en la respuesta.

Un ejemplo completo de una entrada con opciones y control de acceso es el siguiente:

```
<Directory "/var/www/html">
    Options Indexes Multiview
    <RequireAll>
       Require all granted
       Require ip 147.156.0.0/16
    </RequireAll>
</Directory>
```

Que permite que se liste el contenido de los directorios y la negociación del contenido enviado, permitiendo el acceso a todos los ordenadores de la Universidad de Valencia (subred 147.156.0.0/16) y restringiendo el acceso al resto de ordenadores de Internet.

Control de acceso por usuario.

El control de acceso por usuario viene controlado por la directiva *AllowOverride*, la cual debe especificarse siempre dentro de una entrada *Directory*. La directiva *AllowOverride* indica que tipo de directivas están permitidas en los ficheros de control de acceso por usuario. Los valores que puede tomar pueden verse en la siguiente tabla:

Valor	Descripción
`All`	Permitir todas las directivas.
`None`	No permitir ninguna directiva.
`AuthConfig`	Permitir directivas de autentificación de usuarios.
`FileInfo`	Permitir directivas de control del tipo de documentos.
`Indexes`	Permitir directivas de indexado de directorios.
`Limit`	Permitir directivas que controlan el acceso por dirección IP del cliente.
`Options`	Permitir directivas que controlan funcionalidades de los directorios.

Así, la línea:

[31] . Esta opción es ignorada si se utiliza dentro de *Location* y *LocationMatch*.
[32] . Esta opción es ignorada si se utiliza dentro de *Location* y *LocationMatch*.
[33] Server Side Includes (SSI) permite la inclusión de contenidos dinámicos especiales, como la hora de modificación de un archivo o la salida producida por la ejecución de otros programas.

```
AllowOverride Indexes
```

Permite que los ficheros de control dentro de un directorio contengan directivas de indexado de directorios.

En nuestro caso, en el que estamos interesados en el control de acceso por usuario, las entradas *AllowOverride* que nos permiten esto son *All* y *AuthConfig*, de forma que si solo nos interesa permitir que se pueda restringir el acceso a la información a unos usuarios debemos escribir:

```
AllowOverride AuthConfig
```

De esta forma, permitimos que el servidor busque dentro del directorio especificado en la entrada *Directory* dentro de la que se encuentra la directiva *AllowOverride* (o sus subdirectorios), el fichero cuyo nombre se especifica con la directiva *AccessFileName*, cuyo valor por defecto es .htaccess y, en caso de existir dicho fichero, se realice el control de acceso de usuario de acuerdo a las reglas existentes en el fichero.

El fichero de control de acceso por usuario se realiza mediante las directivas *AuthType*, *AuthName*, *AuthUserFile*, *AuthGroupFile* y *Require*.

La directiva *AuthType* indica el tipo de autenticación de usuarios. Existen dos valores posibles *Basic* y *Digest*. *Basic* envía la contraseña entre el cliente y el servidor sin cifrar, por lo que su seguridad depende del canal de comunicación[34], mientras que *Digest* envía la contraseña como un compendio MD5, por lo que nunca es posible capturar la contraseña en texto plano, solo su compendio, pero solo es soportado por algunos clientes Web.

La directiva *AuthName* es una cadena de texto que indica el dominio a utilizar en la autenticación. El dominio sirve para indicar la información debe presentar el cliente al usuario, y para que el cliente sepa que contraseña debe enviar al servidor si la misma ya le ha sido proporcionada para ese dominio.

La directiva *AuthUserFile* indica el nombre del fichero que contiene los nombres de usuarios y sus contraseñas. El fichero con los nombres de usuarios y contraseñas se crea o modifica utilizando el comando */usr/sbin/htpasswd*, que veremos a continuación.

Por su parte, la directiva *AuthGroupFile* indica el nombre del fichero que contiene los nombres de los grupos de usuarios y los usuarios que conforman ese grupo, consistiendo en un fichero de texto plano con líneas con la sintaxis:

```
<nombre del grupo>: <usuario1> <usuario2> ... <usuarioN>
```

Por último, la directiva *Require* indica en este caso los nombres de los usuarios, grupos o todos los usuarios a los que se permite el acceso si proporcionan de forma correcta su contraseña. Las sintaxis para indicar esto son:

[34] Si la comunicación se realiza mediante HTTPS, que veremos con posterioridad, al ser el canal de comunicación seguro la contraseña ira cifrada por el propio canal de comunicación.

```
Require user usuario1 [...usuarioN]
Require group grupo1 [...grupoN]
Require valid-user
```

Donde la opción *Require user* indica los nombres de los usuarios cuyo acceso se permite, obviamente si proporcionan de forma correcta su contraseña; la opción *Require group* indica el nombre de los grupos cuyos usuarios tienen acceso; por último, la opción *Require valid-user* indica cualquier usuario existente en el fichero indicado por *AuthUserFile*.

Un ejemplo de fichero de control del acceso a los recursos mediante usuario es el siguiente:

```
AuthType Basic
AuthName "Acceso Restringido"
AuthUserFile /var/www/users
Require user usuario1 usuario2
```

Mientras que un ejemplo de un fichero que utilice el control de acceso mediante grupos es el siguiente:

```
AuthType Basic
AuthName "Acceso Restringido"
AuthUserFile /var/www/users
AuthGroupFile /var/www/groups
Require group grupo1 grupo2
```

El comando *htpasswd* permite crear o modificar el fichero con los usuarios y contraseñas de autenticación de los usuarios[35]. Su sintaxis básica es:

```
htpasswd -c [-p | -d | -m | -s] <fichero> <usuario>
htpasswd [-p | -d | -m | -s] <fichero> <usuario>
htpasswd -D <fichero> <usuario>
```

Donde la opción *-c* indica que se cree el fichero indicado y, en caso de que exista se destruya y vuelva a crear insertando el usuario indicado, mientras que la opción *-D* indica que se borre el usuario indicado del fichero y si no se especifica ninguna opción se añada el usuario al fichero indicado, el cual debe existir.

Las opciones *-p*, *-d*, *-m* y *-s* indican el modo de cifrado de las contraseñas en el fichero. El modo por defecto es *-d*, que es un cifrado usando *crypt()*[36], mientras que la opción *-p* indica un cifrado en texto plano, esto es, sin cifrar[37], y las opciones *-m* y *-s* indican cifrado utilizando MD5 y SHA respectivamente.

[35] Existe un comando similar *htdigest* para las contraseñas si el método de autenticación es Digest.
[36] Las opciones de cifrado mediante crypt() no es soportado por los servidores Apache en los sistemas operativos Windows, Netware y TPF.
[37] La opción de cifrado en texto plano solo es soportada por los servidores Apache en los sistema operativos Windows, Netware y TPF.

La extensión SSL del servidor Web Apache.

En la configuración del servidor Web hemos visto que las extensiones son ficheros de configuración que se incluyen dentro del fichero de configuración general mediante la orden *Include*.

Entre todas las extensiones existentes, la extensión SSL merece una explicación más detallada, pues es la extensión que permite el acceso al servidor mediante comunicaciones cifradas utilizando el protocolo SSL, lo cual posibilita, por ejemplo, que las contraseñas de acceso de los usuarios a los recursos transcurran por la red de forma segura.

La extensión SSL se encuentra configurada, dentro del directorio de las extensiones, en el fichero *ssl.conf*. El fichero de configuración tiene las siguientes líneas:

```
Listen 147.156.222.65:443
```

Coloca a la escucha el puerto 443, que es el puerto por defecto usado para las conexiones mediante HTTP sobre SSL.

```
SSLPassPhraseDialog  exec:/usr/libexec/httpd-ssl-pass-dialog
```

Indica como debe solicitar el servidor Web la contraseña de acceso a la parte privada del par de claves pública/privada si se encuentra protegida por contraseña. El valor *builtin* indica que se pregunte al usuario en la consola, mientras que el valor *exec:<programa>* indica que se ejecute el programa indicado, al cual el servidor Web le pasa como argumentos el *servidor:puerto* y el tipo de clave (*"RSA"* o *"DSA"*), debiendo el programa devolver la contraseña por la salida estándar.

```
SSLSessionCache         shmcb:/var/cache/mod_ssl/scache(512000)
SSLSessionCacheTimeout  300
```

Indican el tipo de cache de SSL utilizada para poder atender de forma simultánea varias peticiones de un mismo cliente, así como los segundos en que es valida la información de la cache. Los valores posibles de *SSLSessionCache* son *None*, que indica que no se utilice cache, lo que incide en una notable disminución del rendimiento del servidor, *dbm:<fichero de datos>*, que indica que la cache se almacene en un fichero hash en un disco del ordenador o *shmcb<fichero de datos>(bytes)* que indica que la cache se almacene en memoria compartida[38].

```
SSLRandomSeed startup file:/dev/urandom  256
SSLRandomSeed connect builtin
```

SSLRandomSeed indica el tipo de generador de números seudo-aleatorios que se utilizará. Su sintaxis es:

```
SSLRandomSeed <contexto> <fuente> [tamaño]
```

[38] El almacenamiento en memoria compartida no es soportado en todos los sistemas operativos para los que existe el servidor Web Apache.

Donde <*contexto*> indica si es en el arranque (contexto *startup*) o si es al establecer una nueva conexión (contexto *builtin*), mientras que <*fuente*> indica que fuente de número seudo-aleatorios se utilizará. El valor *builtin* indica que se utilice el generador interno del módulo, mientras que *file:<fichero>* indica que se utilice el fichero indicado, que suelen ser los dispositivos */dev/random* o */dev/urandom*, que para acceder al generador de números aleatorios del kernel. Otro valor posible es *exec:<fichero>* que indica que ejecute el fichero indicado el cual devolverá el número aleatorio. Por último, *tamaño* indica el número de bytes que serán utilizados en caso de que el generador devuelva más.

```
SSLCryptoDevice builtin
```

Permite utilizar tarjetas hardware criptográficas instaladas en el sistema.

```
<VirtualHost 147.156.222.65:443>
      ErrorLog logs/ssl_error_log
      TransferLog logs/ssl_access_log
      LogLevel warn
      SSLEngine on
      SSLProtocol all -SSLv2
      SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
      SSLCertificateFile /etc/pki/tls/certs/server.crt
      SSLCertificateKeyFile /etc/pki/tls/private/server.key
</VirtualHost>
```

Por último, se encuentra la definición del servidor virtual que atiende las peticiones siendo destacables los siguientes valores:

SSLEngine {on|off}, que habilita o deshabilita el uso de SSL en el servidor virtual, siendo su valor por defecto off.

SSLProtocol, que indica los protocolos *SSLv3* y/o *TLSv1* que pueden utilizarse. El valor *all* especifica todos los protocolos, y estos pueden añadirse mediante el signo + o eliminarse mediante el signo -, por lo que *all* es equivalente a *+SSLv2 +SSLv3 +TLSv1*, mientras que *all -SSLv2* es equivalente a *+SSLv3 +TLSv1*.

SSLCipherSuite, que indica el tipo de cifrados OpenSSL que pueden utilizarse en la negociación entre el cliente y el servidor[39].

Y *SSLCertificateFile* y *SSLCertificateKeyFile* que indican, respectivamente, la localización de los ficheros con las claves pública y privada del servidor.

Debido al funcionamiento del protocolo SSL, hasta la versión 2.2.12 del servidor Apache no era posible definir más de un servidor virtual que utilizara SSL con la misma dirección IP y puerto. El motivo es que cuando SSL establece la negociación, no conoce el nombre del servidor virtual con el que se deseará establecer la conexión, por lo que negociará siempre con el primer servidor virtual las claves, etc., quedando el resto de servidores virtuales sin posibilidad de acceso.

[39] Una descripción más detallada puede encontrarse en la URL: http://httpd.apache.org/docs/2.0/es/mod/mod_ssl.html#sslciphersuite.

A partir de la versión 2.2.12 del servidor web Apache se añadió la extensión SNI (Server Name Indication), que es una extensión de SSL que permite que un cliente envíe, antes de negociar con el servidor la conexión SSL, el nombre del servidor con el que desea conectarse. De esta forma el servidor conoce previamente el servidor con el que se va a desear establecer la comunicación y, por tanto, enviarle el certificado de ese servidor.

El uso de SNI se activa con la opción:

```
SSLStrictSNIVHostCheck On
```

Qué debe especificarse antes de la definición de cualquier servidor virtual de SSL.

Una vez especificada esa opción, podemos definir un número ilimitado de servidores virtuales sobre SSL utilizando la misma IP y puerto, de igual forma que podíamos hacer en la configuración del servidor sobre el puerto 80, definiendo dentro de cada servidor virtual el camino a sus propios certificados. Así, por ejemplo, podríamos definir dos servidores virtuales sobre SSL como sigue:

```
SSLStrictSNIVHostCheck On

<VirtualHost 147.156.222.65:443>
      NameVirtualHost irtic.uv.es:443
      DocumentRoot /irtic
      ...
      SSLCertificateFile /etc/pki/tls/certs/irtic.crt
      SSLCertificateKeyFile /etc/pki/tls/private/irtic.key
      ...
</VirtualHost>

<VirtualHost 147.156.222.65:443>
      NameVirtualHost www.cdlibre.org:443
      DocumentRoot /cdlibre
      ...
      SSLCertificateFile /etc/pki/tls/certs/cdlibre.crt
      SSLCertificateKeyFile /etc/pki/tls/private/cdlibre.key
      ...
</VirtualHost>
```

El uso de la extensión SNI presenta el problema de que no todos los clientes (navegadores) soportan su uso. En la lista siguiente puede verse de forma sencilla las versiones de los navegadores y de los sistemas operativos que soportan el uso de SNI[40]:

- Internet Explorer 7 o superior en Windows Vista o superior.
- Internet Explorer 8 o superior en Windows XP.
- Firefox 2.0 o superior.
- Opera 8.0 o superior (requiere que la versión 1.1 del protocolo TLS este habilitada).

[40] Información resumida y organizada a partir de la que figura en la Wikipedia.

- Chrome 5.0.342.1 o superior en Mac OS X 10.5.7 o superior.
- Chrome 6 o superior en Windows XP o superior.
- Safari 2.1 o superior en Mac OS X 10.5.6 o superior o Windows Vista o superior.
- Konqueror/KDE 4.7 o superior.

Reenvío de páginas Web.

En determinadas ocasiones, como puede ser cuando una página Web ha cambiado de URL, o bien cuando queremos reenviar una conexión no segura a una conexión segura, es necesario reenviar la llamada realizada, de forma que la petición realizada a la URL original sea enviada a una nueva URL, la cual será la que establezca la conexión con el cliente y atienda su petición[41].

El reenvío de páginas Web puede realizarse utilizando el módulo de Apache *mod_rewrite*, el cual es incluido en la configuración del servidor en el fichero *conf.modules.d/00-ssl.conf*.

El reenvío de conexiones se realiza utilizando las directivas *RewriteEngine*, *RewriteCond* y *RewriteRule*. La directiva:

```
RewriteEngine {On|Off}
```

Habilita (valor *On*) o deshabilita (valor *Off*) el reenvío de conexiones, siendo el valor por defecto Off. Por su parte la directiva:

```
RewriteCond <cadena de texto> <condición>
```

Indica la condición <*condición*> que debe cumplir la cadena de texto <*cadena de texto*> para que sea evaluada la regla de reenvío. La cadena de texto puede contener, además de caracteres, constructores de la forma $N (0<=N<=9) o %N (1<=N<=9) que indican, respectivamente, la referencia a la regla N existente o a la regla N que ha sucedido con anterioridad. Además es posible especificar variables del servidor mediante *%{NOMBRE_VARIABLE}*. Las variables del servidor existentes son:

Cabeceras HTTP			
HTTP_USER_AGENT	HTTP_REFERER	HTTP_COOKIE	HTTP_HOST
HTTP_ACCEPT	HTTP_FORWARDED	HTTP_PROXY_CONNECTION	
Variables internas			
DOCUMENT_ROOT	SERVER_ADMIN	SERVER_NAME	SERVER_ADDR
SERVER_PORT	SERVER_PROTOCOL	SERVER_SOFTWARE	
Conexiones y requerimientos			
REMOTE_ADDR	REMOTE HOST	REMOTE_PORT	REMOTE_USER
REMOTE_IDENT	REQUEST_METHOD	SCRIPT_FILENAME	PATH_INFO

[41] Una información mucho más detallada de la aquí ofrecida, así como ejemplos puede encontrarse en la URL: http://httpd.apache.org/docs/2.0/misc/rewriteguide.html.

QUERY_STRING	AUTH_TYPE		
Fecha y hora del sistema			
TIME_YEAR	TIME_MON	TIME_DAY	TIME_HOUR
TIME_MIN	TIME_SEC	TIME_WDAY	TIME
Otras variables			
API_VERSION	THE_REQUEST	REQUEST_URI	IS_SUBREQ
HTTPS	REQUEST_FILENAME		

Donde las variables corresponden a los nombres similares de las cabeceras MIME, variables del servidor Apache o estructuras de tiempo de los sistemas Linux/UNIX[42]. Por su parte, la condición es una expresión regular de perl. Un reenvío puede tener más de una directiva *RewriteCond*. Por su parte, la directiva:

```
RewriteRule <patrón> <sustitución> [[banderas]]
```

Es la directiva de reenvío propiamente. En ella, <*patrón*> indica el patrón de la cadena que hace que se realice este reenvío, y <*sustitución*> es la URL que reemplaza a la de la llamada. El patrón se una expresión regular en lenguaje perl que se aplica a la URL actual, la cual puede no coincidir con la URL de la llamada si previamente ha sido alterada. Esto es posible, pues puede existir más de una directiva *RewriteRule* , lo que permite, si es necesario, modificar varias veces la URL inicial de llamada.

Por último, *[banderas]* indica una lista de opciones, separadas por comas, que indican alguna acción que queremos que se realice. Existen un gran número de banderas, pero las más comunes son F (prohibido), L (no continuar aplicando directivas *RewriteRule*), y R (devolver al servidor un código 3xx, por defecto 302).

Como hemos indicado, tanto la directiva *RewriteCond* como la directiva *RewriteRule* utilizan expresiones regulares escritas en lenguaje perl. Una breve explicación para poder entender los ejemplos que veremos es la siguiente[43]:

Texto	
.	Un carácter simple.
[caracteres]	Cualquier carácter indicado en “caracteres”.
[^caracteres]	Cualquier carácter no indicado en “caracteres”.
texto1|texto2	Textos alternativos
Cuantificadores	
?	0 o 1 ocurrencias del texto anterior.
*	0 o N ocurrencias del texto anterior (N>0).
+	1 o N ocurrencias del texto anterior (N>1).
Agrupación	
(texto)	Agrupación de textos en uno solo.

[42] Una información más detallada se encuentra en la URL: http://httpd.apache.org/docs/2.2/es/mod/mod_rewrite.html#rewritecond.

[43] Una explicación mucho más detallada de las expresiones regulares en perl puede encontrarse en la URL: http://perldoc.perl.org/perlretut.html

Delimitadores	
^	Comienzo de delimitador de línea.
$	Final de delimitador de línea.
Escape	
\carácter	Escape del significado especial del carácter.

Además, es posible utilizar el carácter *!* como prefijo de un patrón para indicar la negación de ese patrón, de forma que la expresión sea cierta si no se cumple el patrón indicado.

Una vez vistas estas tres directivas, veremos los sencillos ejemplos de reenvío por cambio de la URL y el reenvío de una conexión no segura a una conexión segura.

```
RewriteEngine On
RewriteCond %{HTTP_HOST}   ^dominio_viejo.com$
RewriteRule ^(.*)$         http://dominio_nuevo.com$1 [L,R=301]
RewriteCond %{HTTP_HOST}   ^www.dominio_viejo.com$
RewriteRule ^(.*)$         http://dominio_nuevo.com$1 [L,R=301]
```

Donde podemos ver como si el host solicitado responde al nombre “dominio_viejo.com” es reenviado por la primera regla, mientras que si responde a “www.dominio_viejo.com” es reenviado por la segunda regla.

El reenvío de una conexión no segura a una conexión segura se realiza como:

```
RewriteEngine On
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^(.*)$         https://%{SERVER_NAME}$1 [L,R=303]
```

Donde podemos ver que si el puerto al que se solicita la conexión no es el 443 (HTTPS), esta es reenviada al mismo servidor pero al puerto 443 mediante una URL que empieza con https. Es posible reenviar a segura una parte de la Web indicando estas directivas dentro de una entrada *<Directory>*, tal y como podemos ver en el siguiente ejemplo:

```
<Directory "/var/www/html/directorio">
   RewriteEngine On
   RewriteCond %{SERVER_PORT}    !^443$
   RewriteRule ^(.*)$      https://%{SERVER_NAME}/directorio/$1
[L,R=303]
</Directory>
```

Que reenvía a seguras tan solo las peticiones de páginas dentro del directorio especificado.

Otro ejemplo, un poco más complicado conceptualmente es el siguiente:

```
RewriteEngine On
RewriteCond %{REQUEST_METHOD} GET
RewriteRule ^(.*\.iso)$    ftp://ftp.servidor.com/$1
```

En el cual estamos reenviando las peticiones realizadas (método GET) que correspondan al patrón que indica que están terminadas en .iso a un servidor FTP. Este ejemplo suele utilizarse para descargar a los servidores Web de servir imágenes ISO de CDs/DVDs que van a tardar mucho tiempo en ser ejecutadas y reenviar la petición a un servidor FTP más adecuado para servir este tipo de peticiones[44].

Como comentario final de cualquier reenvío, es necesario tener en cuenta que los valores, opciones, etc., serán los especificados para el servidor que atiende el destino final, por lo que es en el mismo donde deben especificarse todas las condiciones, limitaciones, etc., que deseemos. Así, por ejemplo, si reenviamos una conexión no segura a una conexión segura, y deseamos bloquear el acceso a unas determinadas URLs, este bloqueo deberá realizarse en la configuración del servidor que atiende el puerto 443, y no en el que atiende el puerto 80, pues este tan solo se limitará a realizar el reenvío, sin tener ningún control sobre los permisos, etc., una vez realizado el reenvío.

Ejercicios.

1- Un servidor Web Apache debe alojar las páginas de dos empresas de nombres “empresa1.com” y “empresa2.com”, debiendo mostrar un mensaje de error si se solicita una página que no sea de esas empresas. Con estos requisitos, configurar el servidor Web Apache sabiendo que la dirección IP del ordenador es 147.156.222.65.

2- Un servidor tiene como dirección IP 147.156.222.65. Configurar un servidor Web Apache de forma que escuche conexiones en los puertos 80 y 8080 y limite el acceso al puerto 8080 a los ordenadores de la red 147.156.0.0/16.

3- Disponemos de un servidor Web en la dirección 147.156.222.65 y deseamos crear una zona privada (*/var/www/html/privado*) de forma que solo se permita el acceso a un conjunto de usuarios autenticados. Configurar el servidor Web Apache con estos requisitos.

4- Un servidor, de dirección IP 147.156.222.65, dispone de un servidor Web Apache el cual deseamos configurar para que atienda las peticiones Web en los puertos 80 y 443. Sin embargo, y dado que no poseemos información confidencial, queremos que las peticiones realizadas al puerto 443 sean redirigidas al puerto 80. Configurar el servidor Web con estas condiciones.

5- Un servidor cuya dirección IP es 147.156.222.65, debe reenviar toda conexión que se produzca mediante el puerto 80 al directorio /var/www/html/privado al puerto 443, debiendo además limitar el acceso a dicha información al conjunto de direcciones IP 147.156.0.0/16. Configurar el servidor Web de forma que cumpla estas condiciones.

[44] En el ejemplo se ha enviado a un servidor llamado ftp.servidor.com, pero ese servidor podría ser incluso nuestro propio servidor Web si ejecutará también un servidor de FTP.

Apéndice A: Configuración de los modos de ejecución del servidor.

Como hemos visto con anterioridad, el servidor web Apache tiene dos modos de funcionamiento diferente, modo *prefork* o modo *worker* que se eligen en el momento de compilar el programa. Aunque las opciones de configuración por defecto de ambos modos son validas para la mayoría de usos, puede ser conveniente explicar brevemente las opciones de configuración existente en cada modo.

En el modo *prefork*, un proceso padre lanza procesos hijo para que estos atiendan las peticiones de páginas Web recibidas, procurando que siempre existan algunos procesos hijo en espera para que las solicitudes al cliente no se retrasen debido a la necesidad de lanzar un proceso hijo. Su uso hace que el servidor funcione de forma similar a Apache 1.3 y es el preferible si se utilizan librerías que no admiten hilos y para aislar unas peticiones Web de otras, de forma que un error en una no afecte a las otras peticiones. Si el servidor ha sido compilado en modo *prefork*, las variables que configuran su funcionamiento y sus valores por defecto son:

```
StartServers 8
MinSpareServers 5
MaxSpareServers 20
ServerLimit 256
MaxRequestWorkers 256
MaxConnectionsPerChild 4000
```

Las líneas indican el número inicial de procesos hijo que lanzará el servidor (*StartServers*) y el número mínimo y máximo de procesos hijo que debe tener el servidor en estado de espera de peticiones. Si el número de procesos hijo en estado de espera es menor que *MinSpareServers* el servidor lanzará más procesos hijo, mientras que si es mayor que *MaxSpareServers* el servidor terminará la ejecución de algunos procesos hijo.

Por otra parte, la línea *ServerLimit* indica el número máximo de procesos servidores que pueden especificarse, siendo el valor por defecto de 256[45]. Por su parte las líneas *MaxRequestWorkers* y *MaxConnectionsPerChild* especifican el número máximo de clientes que pueden ser atendidos de forma simultánea y el número máximo de requerimientos que puede atender un proceso hijo antes de terminar y ser sustituido por otro. La razón de esta sustitución de unos procesos por otros procesos es que una permanencia prolongada de los procesos hijos conduciría a una fragmentación y perdida de memoria en el sistema.

En el modo *worker*, el proceso padre lanza procesos hijos, los cuales a su vez ejecutan hilos, permaneciendo uno de ellos a la escucha de peticiones y el resto atendiendo las mismas, sucediendo, como en el otro modo, que existen procesos hijo en espera para no retrasar la atención al cliente. Su principal ventaja es que se utiliza un menor número de recursos que el otro modo para atender el mismo número de clientes. Si el servidor ha sido compilado en modo *worker*, las variables que configuran su funcionamiento y sus valores por defecto son:

[45] Esta línea puede suprimirse si no se requiere la modificación del valor por defecto de 256.

```
ServerLimit 16
StartServers 2
MaxRequestWorkers 150
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25
MaxConnectionsPerChild 0
```

Las líneas indican el número máximo de procesos hijos que lanzará el servidor (*ServerLimit*), el número inicial de procesos hijo que lanzará el servidor (*StartServers*), el número máximo de clientes que pueden ser atendidos de forma simultánea (*MaxRequestWorkers*), el número mínimo y máximo de hilos en espera para atender peticiones (*MinSpareThreads* y *MaxSpareThreads*), el número de hilos que ejecutará cada proceso hijo (*ThreadsPerChild*) y el número máximo de requerimientos que puede atender un proceso hijo antes de terminar y ser sustituido por otro (*MaxConnectionsPerChild*).

En este modo, el valor de *ServerLimit*, que puede ser especificado, indica el número máximo de procesos hijos, que debe ser mayor o igual que *MaxRequestWorkers* dividido por *ThreadsPerChild*[46]. Por otra parte, existe la directiva *ThreadLimit* que indica el número máximo de hilos que puede tener cada proceso hijo y debe ser, obviamente, mayor o igual que el valor de la directiva *ThreadsPerChild*. Si estas dos directivas se utilizan en este modo deben aparecer al principio de todas las directivas del modo.

[46] Esta limitación viene fijada porque si cada proceso hijo puede lanzar X hilos y deseamos atender Y clientes de forma simultánea es obvio que necesitamos un número de hijos mayor o igual a X/Y.