Programación y Administración de Sistemas

Práctica 4. Administración de servidores web: Apache.

Pedro Antonio Gutiérrez

Asignatura "Programación y Administración de Sistemas"
2º Curso Grado en Ingeniería Informática
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

21 de abril de 2015

Índice

`http://www.apache.org/`

Objetivos

Objetivo principal

Conocer a grandes rasgos cómo funciona la WWW a través del servidor web Apache.

Objetivos detallados

- Conocer los protocolos y lenguajes que hacen posible la navegación *web*.
- Conocer las partes del paquete software del servidor Apache: programas, ficheros de configuración, documentación, etc.
- Manejo básico de la configuración de Apache.

Entrega

Entrega y defensa de la práctica

- Durante la práctica guarda en un fichero de texto las órdenes usadas y las directivas de configuración empleadas indicando para qué sirven.
- La entrega de la práctica será una memoria describiendo los pasos necesarios para realizar los ejercicios que hay al final de este documento. Esta memoria deberá presentarse en formato `pdf` y tener una extensión máxima de 15 páginas (incluye capturas para mejorar la descripción). Esta memoria os debe servir para la defensa.
- En el propio fichero de ejemplo que viene con Apache tienes comentarios sobre las directivas en inglés.

Internet con un enfoque descendente

- ¿Qué pasa cuando escribes una dirección y pulsas `intro` en el navegador?
  - URI (identificador uniforme de recurso): "`http://es.wikipedia.org`"
- ¿Qué es un servidor? cliente↔servidor = petición↔respuesta
- Varios servidores intervienen cuando navegáis (Router, DNS, Web, etc.)

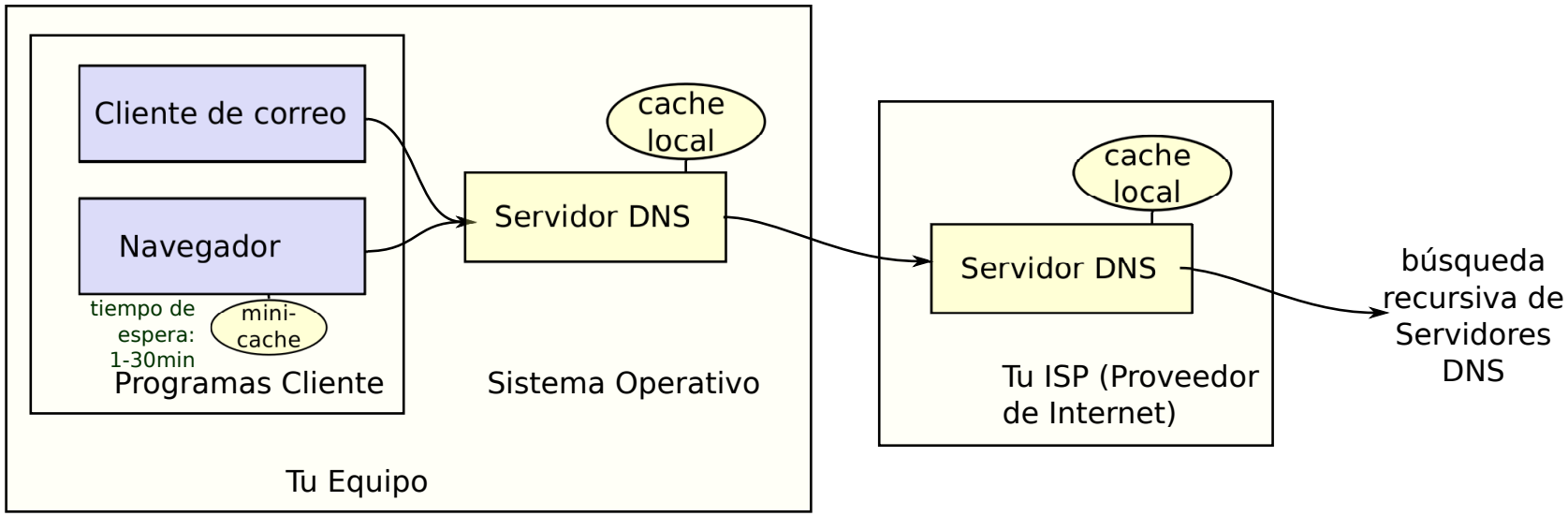
¿Dónde están las cosas?

- A las direcciones IP se les asocian nombres inteligibles.
  ☺ Más fácil de recordar.
  ☺ Más fiable.
- DNS (*Domain Name System*): sistema de nombre de dominio.
- Para averiguarlo, una serie de servidores se pasan mensajes de acuerdo al protocolo DNS.
- Base de datos distribuida y jerárquica.
- Permite asociar distinta información a los nombres de dominio, pero el uso más común es para asociar una Dirección IP.

Identificación de equipos: DNS

- El DNS devuelve la dirección IP asociada a un dominio.
- Muchos dominios pueden estar asociados a una misma IP, así servidores web como Apache permiten alojar varios dominios simultáneamente (*Virtual Hosts*).
- También se pueden asociar múltiples IPs a un dominio para hacer balanceo de carga, por ejemplo, mediante un Round Robin.
- Cliente/Servidor DNS $\rightarrow$ los clientes preguntan al servidor ¿Cuál es la IP de este nombre? y el servidor devuelve la dirección.

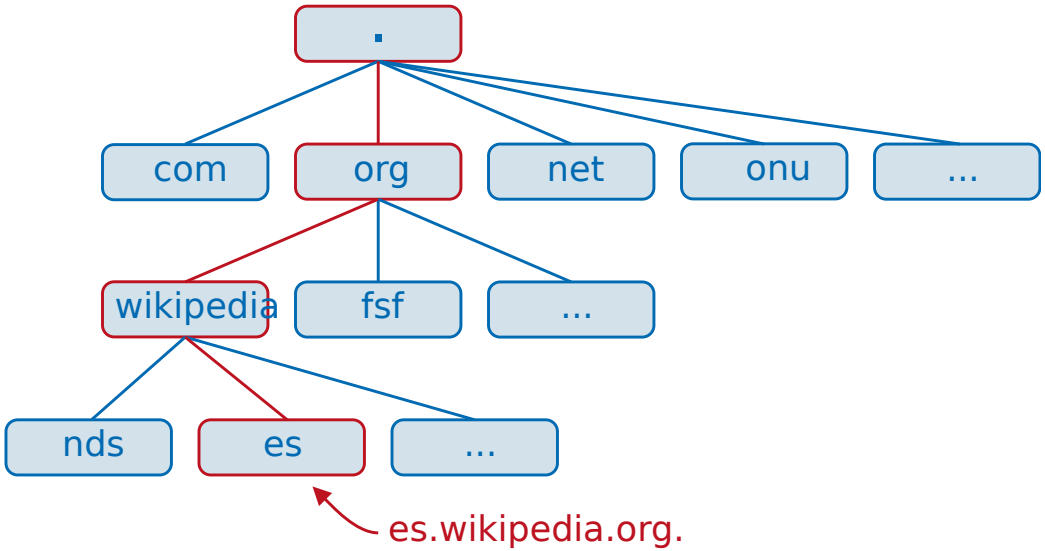
Identificación de equipos: DNS



Identificación de equipos: DNS

- Nombres de dominio: "www.mohamedalid.org", "es.Wikipedia.org", "www.uco.es", "images.google.com".
- Distintas partes (de derecha a izquierda):
  - Dominio de nivel superior: org, es, com...
  - Nombre de la máquina (o *hostname*): mohamedalid, Wikipedia, uco...
  - Subdominios dentro del servidor: www, es, images...

Identificación de equipos: DNS



Identificación de recursos: URI

- URL: estándar del consorcio W3C que define el formato de las cadenas de caracteres que se usan identificar recursos en Internet.
- `protocolo:servidor[:puerto][/pathdelrecurso][?argumentos]`
- *ftp://ftp.download.com/software/prog.exe*
- *http://www.google.com/search?q=kindberg*
- *mailto:joe@anISP.net* (correo con formato más libre)

Metáforas de algunos conceptos

- Protocolo: "comedor universitario". Ejemplo: HTTP
- Dirección IP: "códigos postales + calle + número + piso". Ejemplo: `150.214.110.212`
- Dominio: "la casa de Luis". nombre único y entendible por humanos para no memorizar direcciones IP (entre otras cosas). Ejemplo: `uco.es`

¿Quién muestra las Webs?

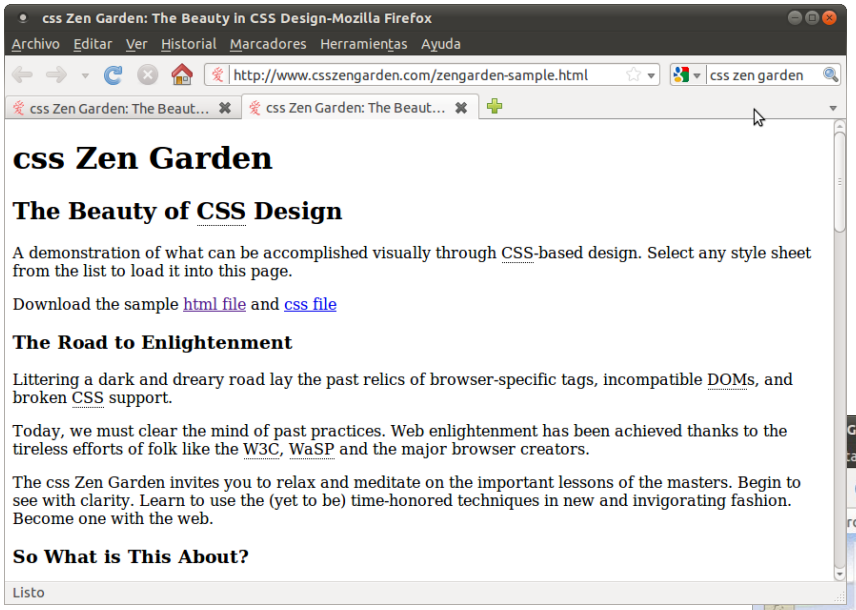
- El SO ya sabe dónde está el servidor web (dirección IP que le pasó el DNS).
- El navegador usa el protocolo HTTP para *pedirle* un recurso (por ejemplo un fichero HTML) al servidor Web, que analiza, y posteriormente hace peticiones adicionales para las imágenes y otros ficheros que formen parte de la página.
- El navegador *renderiza* (muestra) los datos recibidos del servidor tal y como describen el código HTML, CSS y otros lenguajes, y se incorporan las imágenes y otros recursos.

¿Cómo se programa una Web?

Se debe separar contenido y forma:

- El contenido se escribe en HTML o en XHTML (lenguaje extensible de marcado de hipertexto).
- La forma se describe con CSS (hojas de estilo en cascada).
- El XHTML y CSS que describen una web es lo que el servidor envía al navegador (junto con imágenes, vídeos, etc.).
- Las Webs las programan personas desarrollando código directamente o con la ayuda de programas de escritorio o aplicaciones web.

¿Cómo se programa una Web?



Contenido (XHTML)

Contenido+Forma
(XHTML+CSS)

WWW (World Wide Web)

- La World Wide Web (WWW) o Red informática mundial es un sistema de distribución de información basado en hipertexto o hipermedios enlazados y accesibles a través de Internet.
- Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.
- La Web fue creada alrededor de 1989 por el inglés Tim Berners-Lee con la ayuda del belga Robert Cailliau mientras trabajaban en el CERN en Ginebra, Suiza, y publicado en 1992.

Modelo cliente-servidor

- La web funciona siguiendo un modelo cliente-servidor en una red TCP/IP, local o interconectada con las demás redes a través de Internet.
- Cliente: el navegador solicita a un servidor Web el envío de páginas de información. Lo que recibe es un documento de texto HTML, que deberá interpretar.
- Servidor: Atiende las peticiones procedentes de los clientes HTTP.

HTTP (HyperText Transfer Protocol)

- HTTP (HyperText Transfer Protocol): Protocolo de Transferencia de Hipertexto, creado para compartir datos científicos a nivel internacional y de forma instantánea.
- Es el método más común de intercambio de información en la web.
- Descrito en el RFC 1945 (1996) y ampliado y modificado en otros RFCs, el más utilizado es el 2616 (1999) que define la versión 1.1
- Existe una versión segura: HTTPS
- Hoy en día HTTP(S) no se usa sólo para transferir hiper-texto, sino también otros datos (imágenes, vídeo, monitorizar un portaaviones, etc.)...aunque a menudo no sea el protocolo más óptimo para la tarea.

HTTP: Comunicación

- Especificado en el RFC 2616. El contenido de los mensajes son líneas de texto, que contienen ordenes y parámetros con la sintaxis definida.
- Cada transacción es una comunicación distinta (sin estado).
- Tipos de mensaje: petición (*request*) y respuesta (*response*).
- Formato del mensaje:
  - Línea de comienzo: tipo de mensaje (orden HTTP con sus parámetros, *request* o *response*).
  - Líneas de encabezado (si son obligatorias) o cero (si son opcionales), acabadas con un CR-LF (retorno de carro y salto de línea).
  - Separador (CR-LF).
  - Contenido o cuerpo del mensaje.

HTTP: Métodos de petición (request)

- Formato de petición básico: "`método URI versión`".
- El método indica al servidor que debe hacer con el URI [RFC 2396].
- La versión indica la versión del protocolo que el cliente entiende.
- Ejemplo: `GET /index.html HTTP/1.0`
- La versión HTTP/1.0 contempla 3 métodos: `GET`, `HEAD` (como el GET pero sin solicitar el recurso) y `POST` (mandar algo al URI).
- La versión HTTP/1.1 añade: `PUT` (almacenar algo con ese URI), `OPTIONS` (solicitar información sobre URI), `DELETE` (borrar URI).

HTTP: Métodos de respuesta (response)

- Formato de respuesta básico: `versión código_estado texto_explicativo`.
- Ejemplo: `HTTP/1.1 405 Method Not Allowed`
- Ejemplo: `HTTP/1.1 200 Ok`
- Los códigos se clasifican en 5 grupos:
  - Códigos `1xx`: informativos.
  - Códigos `2xx`: éxito de la solicitud.
  - Códigos `3xx`: redireccionar la solicitud.
  - Códigos `4xx`: error generado por el cliente.
  - Códigos `5xx`: error generado por el servidor.

Antes de elegir un servidor HTTP

- Datos de evolución de uso de servidores web a lo largo de los años: `http://news.netcraft.com/archives/category/web-server-survey/`.
- Comparativa de servidores web: `http://en.wikipedia.org/wiki/Comparison_of_web_servers`.

Servidores HTTP I

- NCSA HTTPd: Uno de los primeros, además gratuito. Actualmente el proyecto está abandonado y el sitio web oficial recomienda utilizar Apache.
- Apache: basado en NCSA, es el servidor más utilizado en Internet. Dispone de muchos módulos para ampliar su funcionalidad.
- lighttpd: optimizado para eficiencia. Se usa en sitios como Youtube, Wikipedia, Sourceforge...
- nginx: menos funcional, pero muy eficiente. Suele utilizarse como servidor proxy-inverso para diversos protocolos (HTTP, SMTP, POP3 e IMAP).

Servidores HTTP II

- Cherokee: otro servidor muy eficiente, multi-plataforma y con un panel de administración muy cómodo.
- Internet Information Server (IIS): Desarrollado por Microsoft para sistemas Windows Server.

El servidor Apache I

Apache

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.12 y la noción de sitio virtual.

El servidor Apache II

Ventajas

- Modular
- Software libre
- Multi-plataforma
- Extensible
- Popular (fácil conseguir ayuda/soporte)

Desventajas

Algunos le acusan de no ser eficiente, arquitectura del código obsoleta, etc.

El servidor Apache: módulos

Algunos módulos:

- `mod_ssl` - Comunicaciones Seguras vía TLS.
- `mod_rewrite` - reescritura de direcciones
- `mod_deflate` - Compresión transparente con el algoritmo *deflate* del contenido enviado al cliente.
- `mod_auth_ldap` - Permite autentificar usuarios contra un servidor LDAP.
- `mod_cband` - Control de tráfico y limitador de ancho de banda.
- `mod_perl` - Páginas dinámicas en Perl.
- `mod_php` - Páginas dinámicas en PHP.
- `mod_python` - Páginas dinámicas en Python.
- `mod_ruby` - Páginas dinámicas en Ruby.
- ...

El servidor Apache: LAMP

LAMP

- Linux, el sistema operativo;
- Apache, el servidor *web*;
- MySQL, el gestor de bases de datos;
- Perl, PHP, o Python, los intérpretes para *web* dinámica.

El servidor Apache: Documentación I

- Versión 2.4 de la documentación del Servidor de HTTP Apache: `http://httpd.apache.org/docs/2.4/`
- El Servidor Apache y Programas de Soporte: `http://httpd.apache.org/docs/2.4/programs/`
- Ficheros de configuración: `http://httpd.apache.org/docs/2.4/configuring.html`
- Puertos y direcciones de escucha: `http://httpd.apache.org/docs/2.4/bind.html`
- Iniciar y Parar el servidor Apache: `http://httpd.apache.org/docs/2.4/stopping.html`
- Índice de directivas de configuración: `http://httpd.apache.org/docs/2.4/mod/directives.html`

El servidor Apache: Documentación II

- Índice de Módulos:
  `http://httpd.apache.org/docs/2.4/mod/`
- Secciones de Configuración:
  `http://httpd.apache.org/docs/2.4/sections.html`
- Mapear URLs a partes del sistema de ficheros:
  `http://httpd.apache.org/docs/2.4/urlmapping.html`
- *Virtual hosts*:
  `http://httpd.apache.org/docs/2.4/vhosts/`
- `.htaccess`: `http://httpd.apache.org/docs/2.4/howto/htaccess.html`

Descarga, compilación e instalación I

Utiliza el siguiente *script* para descargar, configurar la instalación e instalar Apache (no requiere privilegios de administración):

```
1  #!/bin/bash
2  mkdir $HOME/tmp
3  cd $HOME/tmp
4
5  # Si por lo que sea la dirección que hay abajo no dispone del fichero, buscar
       otra con Google
6  wget -O httpd-2.4.12.tar.bz2 http://www.eu.apache.org/dist//httpd/httpd-2.4.12.
       tar.bz2
7  tar jxvf httpd-2.4.12.tar.bz2
8
9  # Bajamos y descomprimimos la última versión de las librerías APR
10 wget -O apr-1.5.1.tar.bz2 http://www.eu.apache.org/dist//apr/apr-1.5.1.tar.bz2
11 wget -O apr-util-1.5.4.tar.bz2 http://www.eu.apache.org/dist//apr/apr-util
       -1.5.4.tar.bz2
12 tar jxvf apr-1.5.1.tar.bz2
13 tar jxvf apr-util-1.5.4.tar.bz2
14
15 # Movemos las librerías a la carpeta de apache, para que éste las utilice
16 mv apr-1.5.1 ./httpd-2.4.12/srclib/apr
17 mv apr-util-1.5.4 ./httpd-2.4.12/srclib/apr-util
18
19 # Vamos a la carpeta de apache
20 cd httpd-2.4.12
21
22 # Configurar la compilación, diciendo que el apr lo coja de srclib
23 # prefix va a indicar el directorio de instalación
24 ./configure --prefix=$HOME/httpd --with-included-apr
25
```

Descarga, compilación e instalación II

```
26 # Compilar e instalar en $HOME/httpd
27 make
28 make install
29 make clean
30
31 # Configurar bash para que al hacer logout se maten todos los procesos de Apache
32 if [ ! -f $HOME/.bash_logout ]
33 then
34 echo -e "#!/bin/bash\nkillall -s KILL httpd" > $HOME/.bash_logout
35 elif [ -z "$(cat $HOME/.bash_logout | grep 'killall -s KILL httpd')" ]
36 then
37 echo "killall -s KILL httpd" >> $HOME/.bash_logout
38 fi
39
40 # Borrar el archivo descargado y la carpeta de compilación
41 rm $HOME/tmp/httpd-2.4.12.tar.bz2
42 rm -Rf $HOME/tmp/httpd-2.4.12
43
44 # Borrar las carpetas de las librerías y los archivos descargados
45 rm $HOME/tmp/apr-1.5.1.tar.bz2
46 rm $HOME/tmp/apr-util-1.5.4.tar.bz2
47
48 # Borrar la carpeta temporal
49 rmdir $HOME/tmp
50 cd
```

Ejercicios I

1. Recorre las carpetas del servidor observando qué se guarda en ellas ellas. Necesitarás acceder al archivo `httpd.conf` para modificar la configuración de Apache.
2. Inicia y para el servidor Apache con `apachectl`. Recuerda que tendrás que llamar a `apachectl restart` cada vez que hagas un cambio en la configuración.
   IMPORTANTE: para evitar que Apache siga ejecutándose al salir de nuestra sesión, es obligatorio que añadáis a `$HOME/.bash_logout` la línea `killall -s KILL httpd`.
   Esto ya lo hace el *script* anterior.

Ejercicios II

3. Por defecto, Apache escucha por el puerto 80. Comprueba si esto produce algún error e investiga cuál es el motivo. Modifica el puerto de escucha para que sea el 8080. Comprueba que funciona con un navegador, accediendo a `localhost:8080`.
4. Los archivos que sirve Apache (los que puedo ver desde el cliente) están en una carpeta concreta. Cambia dicha carpeta (*DocumentRoot*), para que sea `$HOME/html-docs/`. Crear la carpeta y copiar dentro un conjunto de archivos `.html` para tu servidor *web*, dándole como nombre a uno de ellos `index.html` e introduciendo en él tu nombre y apellidos. Si no sabes crear ficheros HTML, puedes obtener algunos en `http://www.w3schools.com/html/html_examples.asp`.

Ejercicios III

5. Cambia el nombre de los archivos que se utilizarán por Apache como archivos por defecto cuando solo se especifique el directorio (normalmente es `index.html`, añadir como posibilidad `index.htm`, probar si funciona, ¿qué prioridad se utiliza si tenemos más de un nombre de archivo?).
6. ¿Qué opción de qué directiva es la encargada de permitir mostrar el contenido de un directorio aunque éste no contenga ninguno de los archivos índices explicados en la directiva anterior? Crea un directorio nuevo `$HOME/html-docs/prueba` (es obligatorio especificar una nueva directiva `Directory`), incluye un archivo cualquiera y prueba a utilizar esta opción para prohibir o no el listado de archivos en dicha carpeta. Para probarlo tendrás que acceder a `localhost:8080/prueba`.

Ejercicios IV

¿Cómo podríamos mejorar el aspecto visual del listado? (pistas, `Indexes` y `httpd-autoindex.conf`).

7. Encuentra la directiva que especifica el nombre del servidor e introduce su valor correcto.
8. Encuentra la directiva que especifica el usuario y grupo para el demonio `httpd` (en vuestro caso, ¿sirve para algo cambiarla?).
9. Prueba a hacer un `telnet` al puerto del servidor (`telnet localhost 8080`) y a mandarle mensajes HTML (escribe "`GET / HTTP/1.0`" y pulsa dos veces `intro`). Prueba a escribir HOLA y pulsar dos veces a `Intro`. Explica qué sucede y los códigos de error que devuelve el servidor.

Ejercicios V

10. Encuentra la directiva que se utiliza para la visualización de páginas de error. Modifícala para personalizar el mensaje de error 404 que mostrará el servidor y que sea una cadena del tipo "El URI que has pedido no lo podemos servir". ¿Se podría especificar un fichero `.html` de error?. Modifica el error 501 para que muestre "Metodo no implementado" y comprueba que funciona.
11. Identifica las directivas relacionadas con los archivos de *logs* de Apache. Haz un acceso normal y acceso erróneo (por ejemplo, pidiendo un archivo o directorio que no existe). Comprueba los *logs* y muestra como se han modificado.

Ejercicios VI

12. Redirecciona la dirección `/uco` a `www.uco.es`, de manera que al acceder a `localhost:8080/uco` aparezca la web de la Universidad de Córdoba.
13. Crea un *Host* virtual de manera que cuando un cliente se conecte al servidor usando `localhost:8080` el servidor muestre la carpeta raíz original, y cuando se conecte usando `IPMAQUINA:8080` muestre la subcarpeta `/prueba`. `IPMAQUIMA` es la IP de la máquina en la que estás, puedes averiguarla con `ifconfig`. Haz que tengan ficheros de *log* independientes (`local-access.log`, `local-error.log`, `ip-access.log` y `ip-error.log`).

Ejercicios VII

14. Apache permite el acceso a recursos restringidos mediante la creación de usuarios y grupos que deberán autenticarse antes de acceder a dicho recurso protegido. Se debe:
    - crear los usuarios y contraseñas correspondientes dentro de un archivo llamado `.htpasswd` (para esto, tendrás que utilizar el comando `./htpasswd -c .htpasswd usuario`, incluido en Apache, una vez por cada usuario a añadir y luego copiar el archivo resultante a la carpeta que quieres proteger; ojo, el `-c` indica que el fichero se cree nuevo, por lo que solo debe utilizarse para el primer usuario)
    - e incluir un archivo `.htaccess` (por defecto, aunque estos nombres de archivo pueden modificarse en `httpd.conf`) con los usuarios o grupos de usuarios que tendrán acceso. La sintaxis de ese archivo es la misma que el resto de configuraciones de directorios de Apache.

Ejercicios VIII

Debes crear un directorio, que se llamará `secreto`, de forma que, para acceder a él, habrá que autenticarse. Los usuarios que tendrán acceso a ese directorio serán: `profesor`, `alumno1` y `alumno2`; y la contraseña, para todos será: `quieroentrar`. Explica los pasos realizados para conseguirlo, y el contenido de los archivos creados.
NOTAS: para que el archivo `.htaccess` se interprete por Apache es necesario que, previamente, hayamos activado la directiva `AllowOverride All` sobre el directorio correspondiente (por defecto está a None) en `httpd.conf`. Consultar el ejemplo de autenticación en `http://www.cristalab.com/tutoriales/proteger-carpetas-con-.htaccess-y-.htpasswd-c213l/`.

Ejercicios IX

15. Existe también la posibilidad de permitir o denegar el acceso a diferentes directorios o archivos dependiendo de la dirección IP del cliente. Para ello, disponemos de las directivas `Allow from`, `Deny from` y `Order`, que pueden utilizarse en el archivo de configuración `httpd.conf` o en cada uno de los directorios mediante el archivo `.htaccess`. Prueba esta opción de seguridad para permitir el acceso a nuestro servidor web, únicamente para direcciones IP de la subred de los equipos de sobremesa del laboratorio. Prueba también a denegar el acceso a todas las direcciones IP.
    NOTA: para especificar una subred se debe utilizar la siguiente notación `172.16.215.0/24`, donde 24 es el número de bits de la subred (tres primeras cifras) y la subred sería `172.16.215`. Averigua cuál es la subred del laboratorio con el comando `ifconfig`.

Referencias

Ben Laurie, Peter Laurie.
Apache: The Definitive Guide, 3rd Edition,
O'Reilly Media. Tercera Edición. 2002.

Programación y Administración de Sistemas

Práctica 4. Administración de servidores web: Apache.

Pedro Antonio Gutiérrez

Asignatura "Programación y Administración de Sistemas"
2º Curso Grado en Ingeniería Informática
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

21 de abril de 2015