APACHE

Ayuda en el sitio:

http://httpd.apache.org/docs/2.0/es/mod/core.html

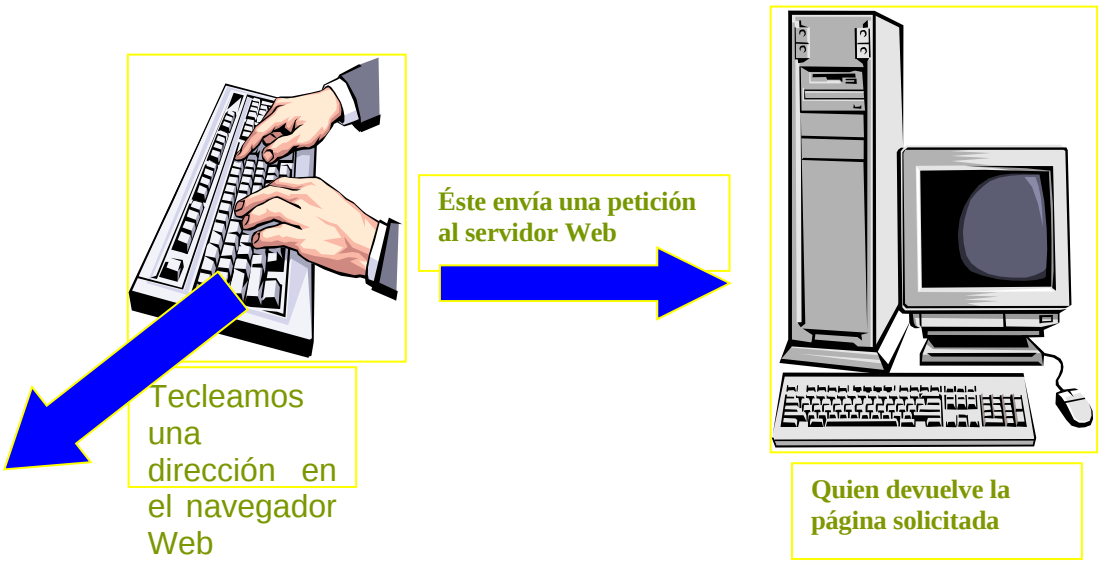
Introducción

Según las estadísticas de Netcraft (http://www.netcraft.com/survey/), Apache es el software servidor web más utilizado en Internet, con más del 60% de la cuota de mercado. Este software gratuito ha sido desarrollado por un grupo de voluntarios que constituyen lo que se conoce como “Apache Group”.
http://apache.org/
Apache corre como un proceso en la máquina servidora, y necesita privilegios de “root” para ser iniciado, ya que ha de abrir un socket en modo escucha sobre un puerto privilegiado 45 . Una vez hecho esto, Apache se deshace de sus privilegios y sigue ejecutándose sin privilegios (normalmente como el usuario “apache”, “httpd” o “nobody”). El proceso padre Apache crea además varios procesos hijos, entre los que reparte las peticiones web recibidas, que se irán procesando en paralelo.

Características generales

- Multiplataforma
- Es un servidor de web conforme al protocolo HTTP/1.1
- Modular: Puede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con la API de programación de módulos, para el desarrollo de módulos específicos.
- Incentiva la realimentación de los usuarios, obteniendo nuevas ideas, informes de fallos y parches para la solución de los mismos.
- Se desarrolla de forma abierta
- Extensible: gracias a ser modular se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor.

Funcionamiento del web



Sintaxis de un URL
`protocolo://dirección[:puerto]/directorio/archivo`

Ejemplos:
http://www.princast.es/
http://195.55.30.17/
http://www.cfacebal.com/index.html
http://localhost:8080/

¿Qué hace un servidor Web cuando se le hace una solicitud?

1. Si el último elemento del URL es un archivo:
    a. Si se incluye una ruta de directorios, lo buscará a partir del indicado en la directiva `DocumentRoot`
2. En el caso de que el último elemento del URL sea un directorio, sin especificar el archivo:

a. Si en dicho directorio existe un archivo `index.html` (o lo que se especifique en `DirectoryIndex`, se devolverá éste
b. Si no existe dicho archivo, y siempre y cuando esté habilitada la opción `Options Indexes` se hará un listado del directorio

HTTP

El protocolo HTTP es el que da vida a Internet, y gracias al cual, los clientes y servidores se pueden comunicar.

El lector, si tiene experiencia en el campo de protocolos, puede pensar que esta es la parte más compleja del Web. Pues bien, este protocolo se diseño con la sencillez en mente, por lo que es de lo más trivial.

El funcionamiento básico es que el cliente estable una conexión TCP con el servidor, hace una petición, el servidor le responde y se cierra la conexión. Para que se haga una idea el lector de la sencillez, en la primera versión ampliamente utilizada del protocolo (1.0), el cliente solo podía invocar tres operaciones en el servidor: GET para pedir una página, HEAD para pedir la cabecera de una página y POST para enviar datos a una URL.

Siento un poco más estrictos, el funcionamiento del protocolo es:

- El cliente envía una petición al servidor. Dicha petición está compuesta por un método a invocar en el servidor (URI) y una versión del protocolo, seguida por un mensaje compatible con MIME con los parámetros de la petición, información del cliente, y un cuerpo opcional con más datos para el servidor. Un ejemplo es:

```
GET  /index.html HTTP/1.1
Accept: text/plain
Accept: text/html
Accept: */*
User-Agent:  Un Agente de Usuario Cualquiera
```

- El servidor responde con una línea de estado, incluyendo la versión del protocolo del mensaje y si la petición tuvo éxito o fracaso, con un código de resultado, seguido de un mensaje compatible con MIME con información del servidor, metainformación (datos a cerca de la información) de la entidad solicitada y un cuerpo opcional con la entidad solicitada. Un ejemplo es:

```
HTTP/1.0 200 OK
Server: MDMA/0.1
```

```
MIME-version: 1.0
Content-type: text/html
Last-Modified: Thu Jul  7 00:25:33 1994
Content-Length: 2003
<title>Página de web del IEEE de Madrid<title>
<hr>
....
<hr>
<h2> Proyectos desarrollados en Internet <h2>
<hr>
```

Pero como todo en el Web, este protocolo (versión 1.1) ya es mucho más potente que en su versión original, y como luego veremos, en total hay ya trece métodos diferentes, además de un conjunto de características nuevas como por ejemplo, el tiempo tras el cual el cliente debe volver a recargar la página.

Los creadores del HTTP 1.1 lo describen como: *"un protocolo de nivel de aplicación orientado a sistemas distribuidos, para la colaboración e hypermedia. Un protocolo genérico, sin estado, orientado a objetos y que puede ser utilizado para muchas aplicaciones, como servidores de nombres y sistemas de gestión de objetos distribuidos, a través de las extensiones de los métodos de petición. Una característica de este protocolo es la negociación de los tipos y representación de los datos, permitiendo que los sistemas no dependan del tipo de datos que se utilicen"*.

Los mismos creador de HTTP 1.0 son conscientes de las limitaciones de escalabilidad y rendimiento del protocolo, por lo que recomiendan que ningún otro servidor lo utilice, y que se utilice de forma única el HTTP 1.1

Los problemas principales que existen en la versión 1.0 son de rendimiento. Esto esta perfectamente documento dentro de http://www.w3.org, y destacamos aquí las conclusiones principales a las que llegaron, pero recomendamos al lector interesado en protocolos que consulte dicho documento:

- Las conexiones del protocolo TCP son lentas de establecer (conexión en tres pasos y ajuste de ventanas de recepción de datos), y como por cada página y cada imagen que haya en la página, ha de establecerse una conexión nueva, la transmisión de datos está ralentizada por el establecimiento de la conexión TCP.

- Una conexión para transmitir 1Kbyte de datos tarda alrededor de 500 ms.

- Tras cerrar una conexión TCP, el puerto del servidor utilizado en dicha conexión, se queda en estado TIME_WAIT un tiempo recomendado de 240 segundos, por lo que un servidor que reciba muchas peticiones

puede agotar todos los puertos TCP (que recordemos son 65535) y dejar al servidor sin posibilidad de enviar ningún tipo de dato. Esto supone un problema de escalabilidad muy importante.

Por lo tanto, a partir de este momento nos centraremos únicamente en el protocolo HTTP 1.1, ya que la versión 1.0 es historia, aunque una historia muy presente que nos puede saltar asi que es mejor tenerlo presente.

Está nueva versión de HTTP está recogida dentro de la RFC 2068 de Enero de 1997, la cual se puede obtener en [1]. Las principales características de esta nueva versión son:

- Conexiones persistentes: ya no se cierra la conexión tras el envío de cada parte de un documento, evitando la sobrecarga del establecimiento de conexiones TCP.

- Varias peticiones simultáneas: un cliente puede realizar varias peticiones utilizando una única conexión, sin esperar a la respuesta del servidor para cada una de ellas.

- Negociación del contenido: se asignan diferentes valores a las características de la comunicación, entre ellos cuanto se puede degradar la calidad de la conexión,

- Nuevos métodos: junto a GET, POST y HEAD aparecen los métodos DELETE para borrar un recurso del servidor asociado al URI de borrado, TRACE para ver que está recibiendo el servidor de lo que él envía, PUT para enviar datos a un recurso asociado a una URI, PATCH para aplicar correcciones en un recurso asociado a una URI, COPY para copiar unos recursos identificados por una URI en otro lugar determinada URI en uno destino determinado, MOVE para mover el recurso identificado por la URI a otro lugar, DELETE para borrar un recurso asociado a una URI, LINK para establecer enlaces entre diferentes recursos, UNLINK para quitar enlaces establecidos previamente por LINK, OPTIONS para que el cliente pueda obtener del servidor sus características, WRAPPED que permite unir varias peticiones y recubrirlas con algún tipo de filtrado (encriptación por ejemplo).

- Nuevo método de autenticación: en la RFC 2069 se describe un nuevo método de autenticación, en el cual las claves de acceso van encriptadas por la red, al contrario de lo que ocurre en HTTP 1.0. Esta RFC aún no se ha unificado con la RFC 2068 para formar la especificación de HTTP 1.1, pero se está en vías de ello.

Junto con estas mejoras, hay muchas ampliaciones en el intercambio de información entre el cliente y el servidor, lo cual ha hecho que la RFC 1954 (Mayo 1996) que describe HTTP 1.0 haya pasado de 60 páginas a 161 página en la RFC 2068 que describe HTTP 1.1. O que el número de autores haya pasado de cuatro a seis, entre ellos Tim Berners-Lee, el "inventor" del web.

El estado de las conexiones sigue sin poderse mantener, a menos que utilicemos mecanismos auxiliares como las cookies (RFC 2109) , pero si bien el no mantener el estado de la conexión tiene sus desventajas, también tiene una importancia fundamental en una red como Internet: algunas de las peticiones HTTP son idempotentes, es decir, que el resultado de invocar un número arbitrario de veces en el servidor, tiene los mismo efectos. Los métodos GET, HEAD, PUT y DELETE tienen esta propiedad. Es importante esta propiedad ya que, para asegurarnos que algo ocurra, y que sea exactamente lo que nosotros queremos que ocurra, podemos invocar estos métodos repetidas veces, de forma que aunque una petición no llegue al servidor, pueda llegar alguna de las peticiones que repiten a esta primera. En la RFC 2068 también se habla de métodos seguros, que son aquellos que su invocación no conlleva ningún riesgo, es decir, que su invocación no tiene efectos laterales en el servidor. Estos métodos son GET y HEAD, que lo único que suponen es una petición de información.

Pero esta nueva versión (HTTP 1.1) es un puente hacia lo que en realidad se quiere imponer en Internet: HTTP-NG (HTTP Next Generation). Este nuevo protocolo pretende cubrir una gran cantidad de nuevas funcionalidades, entre la que destaca el comercio electrónico. Sus criterios de diseño han sido:

- Simplicidad: no se debe abandonar el criterio introducido en HTTP 1.0, que las cosas habituales sean sencillas, de forma que sea fácil implementar el protocolo

- Rendimiento: debe ser eficiente trasmitiendo objetos en redes de comunicaciones

- Asíncrono: las peticiones desde los clientes han de poderse hacer en paralelo a través de una única conexión

- Seguridad: los objetos que se transmiten deben estar encriptados, sin forzar ninguna política de seguridad en particular.

- Autenticación: se debe poder autenticar a las dos partes de la conexión, así como a cualquier intermediario. Pagos en línea: el protocolo debe soportar la realización de pagos en línea

- Servidores intermediarios: se debe soportar la comunicación entre servidores, para el mantenimiento de cachés, espejos de datos e intermediarios de comunicación (proxys).

- Visualización obligatoria: se debe poder obligar al cliente a mostrar ciertos datos acerca del objeto que se transmite, como el autor del objeto, el copyright y la licencia.

- Información de registro: la información de registro (logs) ha de poder ser enviada entre diferentes servidores.

- Requerimientos de red: el protocolo debe trabajar de forma independiente de la capa de transporte de la que disponga, aunque debe funcionar especialmente bien con TCP, al ser el protocolo más utilizado en Internet.

Como puede observar el lector, los avances en HTTP 1.1 (y más aún en HTTPng), son muy importantes, y un servidor como Apache (cuya cuota de mercado es superior al 45%) en su versión 1.2 ya tiene incluido este nuevo protocolo. Y todos los clientes de WWW lo soportarán, dando una nueva vitalidad al día a día en Internet.

En Apache 3.0, que aún está en un futuro lejano, el principal y único objetivo a día de hoy es dar soporte a HTTP-NG. Si Apache implementa este protocolo de forma automática puede alcanzar al 60% del mercado, lo que le daría un impulso definitivo.

```
El protocolo HTTP es el que da vida a Internet, y gracias
al cual, los clientes y servidores se pueden comunicar.
```

El lector, si tiene experiencia en el campo de protocolos, puede pensar que esta es la parte más compleja del Web. Pues bien, este protocolo se diseño con la sencillez en mente, por lo que es de lo más trivial.

El funcionamiento básico es que el cliente estable una conexión TCP con el servidor, hace una petición, el servidor le responde y se cierra la conexión. Para que se haga una idea el lector de la sencillez, en la primera versión ampliamente utilizada del protocolo (1.0), el cliente solo podía invocar tres operaciones en el servidor: GET para pedir una página, HEAD para pedir la cabecera de una página y POST para enviar datos a una URL.

Siento un poco más estrictos, el funcionamiento del protocolo es:

- El cliente envía una petición al servidor. Dicha petición está compuesta por un método a invocar en el servidor (URI) y una versión del protocolo,

seguida por un mensaje compatible con MIME con los parámetros de la petición, información del cliente, y un cuerpo opcional con más datos para el servidor. Un ejemplo es:

```
GET  /index.html HTTP/1.1
Accept: text/plain
Accept: text/html
Accept: */*
User-Agent:  Un Agente de Usuario Cualquiera
```

- El servidor responde con una línea de estado, incluyendo la versión del protocolo del mensaje y si la petición tuvo éxito o fracaso, con un código de resultado, seguido de un mensaje compatible con MIME con información del servidor, metainformación (datos a cerca de la información) de la entidad solicitada y un cuerpo opcional con la entidad solicitada. Un ejemplo es:

```
HTTP/1.0 200 OK
Server: MDMA/0.1
MIME-version: 1.0
Content-type: text/html
Last-Modified: Thu Jul  7 00:25:33 1994
Content-Length: 2003
<title>Página de web del IEEE de Madrid<title>
<hr>
....
<hr>
<h2> Proyectos desarrollados en Internet <h2>
<hr>
```

Pero como todo en el Web, este protocolo (versión 1.1) ya es mucho más potente que en su versión original, y como luego veremos, en total hay ya trece métodos diferentes, además de un conjunto de características nuevas como por ejemplo, el tiempo tras el cual el cliente debe volver a recargar la página.

Los creadores del HTTP 1.1 lo describen como: *"un protocolo de nivel de aplicación orientado a sistemas distribuidos, para la colaboración e hypermedia. Un protocolo genérico, sin estado, orientado a objetos y que puede ser utilizado para muchas aplicaciones, como servidores de nombres y sistemas de gestión de objetos distribuidos, a través de las extensiones de los métodos de petición. Una característica de este protocolo es la negociación de los tipos y representación de los datos, permitiendo que los sistemas no dependan del tipo de datos que se utilicen"*.

Los mismos creador de HTTP 1.0 son conscientes de las limitaciones de escalabilidad y rendimiento del protocolo, por lo que recomiendan que ningún otro servidor lo utilice, y que se utilice de forma única el HTTP 1.1

Los problemas principales que existen en la versión 1.0 son de rendimiento. Esto esta perfectamente documento dentro de http://www.w3.org, y destacamos aquí las conclusiones principales a las que llegaron, pero recomendamos al lector interesado en protocolos que consulte dicho documento:

- Las conexiones del protocolo TCP son lentas de establecer (conexión en tres pasos y ajuste de ventanas de recepción de datos), y como por cada página y cada imagen que haya en la página, ha de establecerse una conexión nueva, la transmisión de datos está ralentizada por el establecimiento de la conexión TCP.

- Una conexión para transmitir 1Kbyte de datos tarda alrededor de 500 ms.

- Tras cerrar una conexión TCP, el puerto del servidor utilizado en dicha conexión, se queda en estado TIME_WAIT un tiempo recomendado de 240 segundos, por lo que un servidor que reciba muchas peticiones puede agotar todos los puertos TCP (que recordemos son 65535) y dejar al servidor sin posibilidad de enviar ningún tipo de dato. Esto supone un problema de escalabilidad muy importante.

Por lo tanto, a partir de este momento nos centraremos únicamente en el protocolo HTTP 1.1, ya que la versión 1.0 es historia, aunque una historia muy presente que nos puede saltar asi que es mejor tenerlo presente.

Está nueva versión de HTTP está recogida dentro de la RFC 2068 de Enero de 1997, la cual se puede obtener en [1]. Las principales características de esta nueva versión son:

- Conexiones persistentes: ya no se cierra la conexión tras el envío de cada parte de un documento, evitando la sobrecarga del establecimiento de conexiones TCP.

- Varias peticiones simultáneas: un cliente puede realizar varias peticiones utilizando una única conexión, sin esperar a la respuesta del servidor para cada una de ellas.

- Negociación del contenido: se asignan diferentes valores a las características de la comunicación, entre ellos cuanto se puede degradar la calidad de la conexión,

- Nuevos métodos: junto a GET, POST y HEAD aparecen los métodos DELETE para borrar un recurso del servidor asociado al URI de borrado, TRACE para ver que está recibiendo el servidor de lo que él envía, PUT

para enviar datos a un recurso asociado a una URI, PATCH para aplicar correcciones en un recurso asociado a una URI, COPY para copiar unos recursos identificados por una URI en otro lugar determinada URI en uno destino determinado, MOVE para mover el recurso identificado por la URI a otro lugar, DELETE para borrar un recurso asociado a una URI, LINK para establecer enlaces entre diferentes recursos, UNLINK para quitar enlaces establecidos previamente por LINK, OPTIONS para que el cliente pueda obtener del servidor sus características, WRAPPED que permite unir varias peticiones y recubrirlas con algún tipo de filtrado (encriptación por ejemplo).

- Nuevo método de autenticación: en la RFC 2069 se describe un nuevo método de autenticación, en el cual las claves de acceso van encriptadas por la red, al contrario de lo que ocurre en HTTP 1.0. Esta RFC aún no se ha unificado con la RFC 2068 para formar la especificación de HTTP 1.1, pero se está en vías de ello.

Junto con estas mejoras, hay muchas ampliaciones en el intercambio de información entre el cliente y el servidor, lo cual ha hecho que la RFC 1954 (Mayo 1996) que describe HTTP 1.0 haya pasado de 60 páginas a 161 página en la RFC 2068 que describe HTTP 1.1. O que el número de autores haya pasado de cuatro a seis, entre ellos Tim Berners-Lee, el "inventor" del web.

El estado de las conexiones sigue sin poderse mantener, a menos que utilicemos mecanismos auxiliares como las cookies (RFC 2109) , pero si bien el no mantener el estado de la conexión tiene sus desventajas, también tiene una importancia fundamental en una red como Internet: algunas de las peticiones HTTP son idempotentes, es decir, que el resultado de invocar un número arbitrario de veces en el servidor, tiene los mismo efectos. Los métodos GET, HEAD, PUT y DELETE tienen esta propiedad. Es importante esta propiedad ya que, para asegurarnos que algo ocurra, y que sea exactamente lo que nosotros queremos que ocurra, podemos invocar estos métodos repetidas veces, de forma que aunque una petición no llegue al servidor, pueda llegar alguna de las peticiones que repiten a esta primera. En la RFC 2068 también se habla de métodos seguros, que son aquellos que su invocación no conlleva ningún riesgo, es decir, que su invocación no tiene efectos laterales en el servidor. Estos métodos son GET y HEAD, que lo único que suponen es una petición de información.

Pero esta nueva versión (HTTP 1.1) es un puente hacia lo que en realidad se quiere imponer en Internet: HTTP-NG (HTTP Next Generation). Este nuevo protocolo pretende cubrir una gran cantidad de nuevas funcionalidades, entre la que destaca el comercio electrónico. Sus criterios de diseño han sido:

- Simplicidad: no se debe abandonar el criterio introducido en HTTP 1.0, que las cosas habituales sean sencillas, de forma que sea fácil implementar el protocolo

- Rendimiento: debe ser eficiente trasmitiendo objetos en redes de comunicaciones

- Asíncrono: las peticiones desde los clientes han de poderse hacer en paralelo a través de una única conexión

- Seguridad: los objetos que se transmiten deben estar encriptados, sin forzar ninguna política de seguridad en particular.

- Autenticación: se debe poder autenticar a las dos partes de la conexión, así como a cualquier intermediario. Pagos en línea: el protocolo debe soportar la realización de pagos en línea

- Servidores intermediarios: se debe soportar la comunicación entre servidores, para el mantenimiento de cachés, espejos de datos e intermediarios de comunicación (proxys).

- Visualización obligatoria: se debe poder obligar al cliente a mostrar ciertos datos acerca del objeto que se transmite, como el autor del objeto, el copyright y la licencia.

- Información de registro: la información de registro (logs) ha de poder ser enviada entre diferentes servidores.

- Requerimientos de red: el protocolo debe trabajar de forma independiente de la capa de transporte de la que disponga, aunque debe funcionar especialmente bien con TCP, al ser el protocolo más utilizado en Internet.

Como puede observar el lector, los avances en HTTP 1.1 (y más aún en HTTPng), son muy importantes, y un servidor como Apache (cuya cuota de mercado es superior al 45%) en su versión 1.2 ya tiene incluido este nuevo protocolo. Y todos los clientes de WWW lo soportarán, dando una nueva vitalidad al día a día en Internet.

En Apache 3.0, que aún está en un futuro lejano, el principal y único objetivo a día de hoy es dar soporte a HTTP-NG. Si Apache implementa este protocolo de forma automática puede alcanzar al 60% del mercado, lo que le daría un impulso definitivo.

Configuración

Todas las directivas de funcionamiento se incluyen en el archivo **`httpd.conf`**
Dependiendo de la versión se puede encontrar en el directorio /**`etc`** o en **`/etc/http/conf`**.
Está constituido por una serie de directivas que dan al servidor instrucciones para su funcionamiento. Estas directivas se agrupan en tres secciones básicas:

Secciones:

- Directivas globales, que controlan el funcionamiento del servidor como un todo (el "entorno global").
- Directivas para el servidor principal, que definen los parámetros del servidor principal., o por defecto, que responde a las solicitudes que no son manejadas por servidores virtuales.
- Directivas para las máquinas (*hosts*) virtuales, configuración de los servidores virtuales, que permiten enviar las peticiones web a diferentes direcciones IP o nombres de host, pero manejándolas en realidad con el mismo proceso servidor.

El archivo de configuración sigue las siguientes reglas:

- # comentario
- **** continuación de línea
- Las líneas en blanco se ignoran.

Apache tiene varios directivas de bloque que limitan la aplicación de otro directiva , por ejemplo: configuración de módulos, directorios, sitios virtuales.

Su sintaxis es:

```
<directiva>
     opciones especificas
</directiva>
```

Directivas del entorno global

Son las que afectan al funcionamiento general de Apache; por ejemplo: el número de peticiones concurrentes a las que puede atender, dónde se ubican los archivos de configuración, el puerto en el que "escucha", etcétera
Entre las directivas de esta sección tenemos:

ServerTokens

ServerTokens OS

Contestación que se envía a los clientes en la cabecera de los campos, posibles valores:

```
#ServerTokens Prod[uctOnly]  Server: Apache
#ServerTokens Min : Server: Apache/1.3.0
#ServerTokens OS  Server: Apache/1.3.0 (Unix)
#ServerTokens Full     Server: Apache/1.3.0 (Unix) PHP/3.0 MyMod/1.2
```

ServerRoot

ServerRoot "/etc/httpd"

Raíz del árbol de directorios bajo la que se sitúan los archivos de configuración, error y log.

En el siguiente cuadro vemos la localización de los archivos de configuración.

```
[root@server1 httpd]# pwd
/etc/httpd
[root@server1 httpd]# ls -l
total 28
drwxr-xr-x 2 root root 4096 ago 22 11:47 conf
drwxr-xr-x 2 root root 4096 abr 19 13:31 conf.d
lrwxrwxrwx 1 root root   19 abr 19 13:25 logs -> ../../var/log/httpd
lrwxrwxrwx 1 root root   27 abr 19 13:25 modules ->
../../usr/lib/httpd/modules
lrwxrwxrwx 1 root root   13 abr 19 13:25 run -> ../../var/run
[root@server1 httpd]#
```

PidFile

`PidFile /usr/local/apache/httpd.pid`

Archivo en el que se almacena el identificador de proceso del servidor cuando el servidor arranca. Este identificador es el que hay que utilizar para detener el servidor si no se utiliza la orden "`apachectl stop`" o "`service httpd stop`"

El comando `apachectl` permite administrar a httpd

TimeOut

`TimeOut 300`

Numero de segundos desde que ser recibe la petición y se envía la señal de time out.

KeepAlive

`KeepAlive off`

Indica si se permiten o no conexiones que se reintentan consecutivamente, mas de una petición por conexión, o sea persistentes.

MaxKeepAliveRequests

`MaxKeepAliveRequests 100`

Indica el numero máximo de peticiones por conexión en una conexión persistente, un valor cero indica que se permite peticiones ilimitadas

KeepAliveTimeout

`KeepAliveTimeout 15`

Indica el numero de segundos a esperar entre dos peticiones del mismo cliente en la misma conexión,

Modulo MPM

El Módulo Multi-proceso (MPM) implementa un servidor híbrido multi-proceso multi-hilos. Usando los hilos para atender las demandas de los clientes, puede servir un número grande de demandas con menos recursos del sistema.

```
<IfModule prefork.c>
    StartServers       8
    MinSpareServers    5
    MaxSpareServers   20
    ServerLimit      256
    MaxClients       256
    MaxRequestsPerChild  4000
</IfModule>
```

Valores:
StartServers: numero de procesos hijos del servidor creados al inicio
MinSpareServers: El número mínimo de hilos ociosos
MaxSpareServers: El número máximo de hilos ociosos
ServerLimit: maximum value for MaxClients for the lifetime of the server
MaxClients: limite de servidores que se pueden ejecutar, o sea la cantidad de conexiones que puede tener simultáneamente.
MaxRequestsPerChild: numero máximo de peticiones que cada proceso hijo puede atender antes de morir.

Listen

```
Listen *:80
```
Permite escuchar en otro puerto, especificar la dirección IP.

Dynamic Shared Object (DSO) Support

Módulos que se pueden cargar dinámicamente.
`LoadModule access_module modules/mod_access.so`

Include

Include conf.d/*.conf

Se incluyen todos los archivos contenidos en el directorio. Agregando a la configuración del servidor.
Por ejemplo configuración de php:

```
cat php.conf
# PHP is an HTML-embedded scripting language which attempts to make it
# easy for developers to write dynamically generated webpages.
LoadModule php4_module modules/libphp4.so
# Cause the PHP interpreter to handle files with a .php extension.
AddType application/x-httpd-php .php
# Add index.php to the list of files that will be served as directory
# indexes.
DirectoryIndex index.php
```

ExtendedStatus

`ExtendedStatus On`
Indica si Apache generará información de estado completa o simple.

User

`User apache`
Usuario con el que se ejecutará el demonio.

Group

`Group apache`
Grupo con el que se ejecutará el demonio.

Directivas del servidor principal

Section 2: 'Main' server configuration

ServerAdmin

`ServerAdmin root@localhost`

Mail del usuario a quien se informará si surge un error.

ServerName

`ServerName www.dominio.com.uy:80`

Nombre del servidor y puerto

UseCanonicalName

`UseCanonicalName on`

Determina como construye una referencia, si esta en off utiliza los datos dados por el cliente, si esta en on utiliza ServerName

DocumentRoot

`DocumentRoot "/var/www/html"`

Directorio que contendrá los documentos que se ofrecen a los clientes. Por defecto todas las solicitudes se toman desde el directorio aunque se pueden utilizar enlaces simbólicos para apuntar a otros directorios

Directory

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

La directiva **`Directory`** se utiliza para definir opciones específicas que se aplican al directorio indicado. No puede utilizarse con caminos relativos, pero permite utilizar expresiones regulares para indicar el directorio. Lo habitual es configurar por defecto unos permisos muy restrictivos y posteriormente habilitar el acceso y permisos más amplios en directorios específicos. Por ejemplo, el bloque anterior permite únicamente seguir enlaces simbólicos directiva **`Options FollowSymLinks`**, y mediante la directiva **`AllowOverride None`** impide que se puedan modificar los permisos establecidos en este archivo

(**`httpd.conf`**), por otros indicados en archivos .**`htaccess`** (particulares de cada directorio del servidor). A partir de aquí habilitaremos el acceso a la parte del servidor que nos interese y a todos, o a algunos usuarios determinados.

La ruta especificada en esta directiva debe coincidir con la raíz del árbol de documentos y las opciones que se especifican a continuación hacen referencia a este directorio y sus subdirectorios.

`Options Indexes FollowSymLinks Multiviews`
Al activar la opción “Indexes” (ordenación de directorios) el servidor permite que si se solicita un URL que corresponde a un directorio y no hay página por defecto para mostrar (por ejemplo: index.html) se devuelva un listado de los archivos del directorio. Si esta opción está activada hay que asegurarse que los directorios no contienen archivos sensibles: listas de control de acceso, archivos de configuración o bases de datos como “.**`htpasswd`**” y “.**`htaccess`**”. A no ser que se desee ofrecer navegación por archivos, es más seguro no activar esta opción. La opción puede desactivarse anteponiendo un signo “ – “:
`Options –Indexes, o simplemente no colocandola`

Tener en cuenta que una vez activada para un directorio la opción permanece activa para todos los subdirectorios del directorio a no ser que se modifique.

`FollowSymLinks`
La opción “**`FollowSymLinks`**” permite que puedan seguirse los enlaces simbólicos en este directorio. Esto tiene serias implicaciones de seguridad porque los usuarios locales pueden, inadvertidamente, (o incluso de forma maliciosa) vincular a archivos de sistema internos y, por tanto, “romper la barrera”, permitiendo a usuarios remotos saltar por encima de la barrera virtual que separa el espacio web de la jerarquía del sistema de archivos principal. En principio esta opción no debería activarse. Por último, la opción “**`Multiviews`**” permite negociar los contenidos con el navegador para elegir la mejor representación de acuerdo con sus preferencias (idioma, conjunto de caracteres, codificación, …).

Limit

El propósito <Limit> es restringir el efecto de los controles de acceso a los métodos HTTP que se especifiquen. Para los demás métodos, las restricciones de acceso que no estén incluidas en <Limit> **no tendrán efecto**. Los siguientes ejemplos aplican el control de acceso solo a los métodos POST, PUT, y DELETE, no afectando al resto de métodos:

```
<Limit POST PUT DELETE>
     Require valid-user
```

```
</Limit>
```

Los métodos incluidos en la lista pueden ser los siguientes: GET, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, y UNLOCK. Los nombres de los métodos distinguen mayúsculas de minúsculas. Si usa GET también se restringirán las peticiones HEAD. El método TRACE no puede limitarse.

LimitExcept

Es mejor usar una sección **<LimitExcept>** en lugar de una sección **<Limit>** cuando se quiere restringir el acceso, porque una sección **<LimitExcept>** protege contra métodos arbitrarios.
Restringe los controles de acceso a todos los métodos HTTP excepto a los que se especifiquen.
<LimitExcept> y </LimitExcept> se usan para englobar un grupo de directivas de control de acceso que se aplicarán a cualquier método de acceso HTTP no especificado en los argumentos; es lo contrario a lo que hace una sección <Limit> y puede usarse para controlar tanto métodos estándar como no estándar o métodos no reconocidos.

```
<LimitExcept OPTIONS>
     Require valid-user
</LimitExcept>
```

`Order allow,deny`
Las dos directivas permiten realizar control del acceso al servidor:

Determina el orden en que se evalúan los derechos de acceso. Hay tres opciones:
- **`allow,deny`**
- **`deny,allow`**
- **`mutual-failure`**: especifica que una conexión debe pasar las directivas **`allow`** y **`deny`**

La directiva "**`allow`**" controla qué hosts (si hay alguno) pueden conectarse y ofrece tres opciones: **`all`**, **`none`** o **`list`** (donde **`list`** es una lista de hosts autorizados).
La directiva "**`deny`**" controla qué hosts (si hay alguno) no pueden conectarse y ofrece tres opciones: **`all`**, **`none`** o **`list`** (donde **`list`** es una lista de hosts no autorizados).
`Allow from all` Todo el mundo puede acceder al directorio especificado. Puede también habilitarse/denegarse acceso restringido indicando en lugar de **`all`**, nombres de dominio o direcciones IP. Pueden especificarse múltiples nombres o direcciones separados por espacios en blanco.

```
order allow,deny          #orden de evaluacion, primero
allow
allow from 123.156.3.56   #solo este hosts
deny from all             #se niega a todos los demás hosts
</Directory>
```

Match	Allow,Deny result	Deny,Allow result
Match Allow only	Request allowed	Request allowed
Match Deny only	Request denied	Request denied
No match	Default to second directive: Denied	Default to second directive: Allowed
Match both Allow & Deny	Final match controls: Denied	Final match controls: Allowed

In the following example, all hosts in the apache.org domain are allowed access; all other hosts are denied access.

```
Order Deny,Allow
Deny from all
Allow from apache.org
```

EJEMPLOS con dominios

```
Allow from apache.org
Allow from .net example.edu
```

Ejemplo con IP

```
Allow from 10.1.2.3
Allow from 192.168.1.104 192.168.1.205
Ejemplo
Allow from 10.1
Allow from 10 172.20 192.168.2
Ejemplo
Allow from 10.1.0.0/255.255.0.0
Ejemplo
Allow from 10.1.0.0/16
```

Permisos de acceso

`AllowOverride None`
Controla que opciones pueden modificar los .**`htaccess`** de los directorios de usuario. Puede impedirse totalmente la modificación con "**`None`**", permitirse totalmente con "**`All`**", o de forma parcial utilizando otros parámetros. (Consultar la documentación en caso necesario).

Configuración de acceso por usuarios a un directorio

Modulo acceso usuarios

```
<IfModule mod_userdir.c>
    UserDir "disabled"
    UserDir "enabled" solange
    UserDir /home/*/www
</IfModule>
```

Ruta que se añade al directorio home del usuario si se recibe una solicitud del tipo **~usuario**. Cuando por ejemplo ingresamos la dirección en el navegador: **http://server.com.uy/~solange** el servidor accederá al directorio: `/home/solange/www/index.html`

Se permitirá el ingreso luego de certificarse como usuario e ingresando la contraseña, ver mas adelante creación del archivo de usuarios

Página inicial

```
DirectoryIndex index.html
```

Establece el nombre de los archivos que se utilizarán como páginas por defecto cuando el URL sólo indica directorios, pero no especifica ningún archivo. Pueden indicarse varios nombres de archivo separados por espacios.
Si el archivo especificado por la directiva **DirectoryIndex** no existe el comportamiento del servidor variará dependiendo de que esté permitido o no el listado del directorio.

Archivo .htacces

La directivas siguientes se utilizan para restringir el acceso a los directorios mediante autentificación HTTP básica, el archivo contiene las directivas de acceso, es una alternativa a configurarlo en el archivo http.conf

AccessFileName .htacces
Nombre del archivo para mirar información de control de acceso en cada directorio.

```
< Files ~"^\.ht">
Order allow deny
Deny from all
</Files>
```

Impide que los .**htaccess** pueden ser vistos por los usuarios que acceden.

Control de acceso por usuario

Control de acceso a directorios mediante autentificación de usuarios El acceso a los directorios puede restringirse de dos formas: utilizando el nombre de dominio o la dirección IP del ordenador donde se ejecuta el cliente, o solicitando un nombre de usuario y una contraseña. Para el primer tipo de restricción ya hemos comentado las directivas necesarias, así es que en este apartado vamos a ver como llevar a cabo la segunda posibilidad.
Para restringir el acceso mediante nombre de usuario y contraseña serán necesarios dos pasos: en primer lugar crear un archivo que contenga los nombres de los usuarios autorizados (que no tiene porque coincidir con usuarios del sistema), y en segundo lugar indicar al servidor que recursos están protegidos y cuáles son los usuarios que pueden acceder a ellos. Para ello se utilizarán la herramienta **`htpasswd`** .

Creación de la base de datos de usuarios

Necesitamos crear una lista de usuarios con sus contraseñas asociadas. Por seguridad esta lista NO DEBE ESTAR BAJO EL DIRECTORIO RAÍZ. El formato es similar al archivo de contraseñas de Unix, con el nombre de usuario separado de la contraseña por dos puntos. Cuando los usuarios piden acceso a un directorio web protegido, el servidor les pide el nombre de usuario y la contraseña. Después compara esos valores con los que tiene almacenados en el usuario de contraseñas (le llamaremos `archivo_de_passwords`).

Comando htpasswd

Como las contraseñas se almacenan cifradas, este archivo no se edita directamente, sino que se crea y se gestionan sus usuarios mediante el programa htpasswd (/usr/local/apache/bin).

`htpasswd [-c] archivo_de_passwords nombre_usuario`

La opción –c se utiliza para crear el archivo cuando se introducen los datos del primer usuario. El programa pedirá la contraseña del usuario y su confirmación. Puede utilizarse también para modificar las contraseñas de usuarios ya incluidos en el archivo.

Ejemplo creación de una cuenta de usuario solange

```
htpasswd /usr/local/apache/passwd/passwords solange
New password:
Re-type new password:
Updating password for user solange
```

[root@server1 httpd]# cat /usr/local/apache/passwd/password
solange:pTCePQJYjAv4I

Habilitar grupo

Se puede implementar la seguridad a nivel de grupo, si necesitamos habilitar a mas de un usuario. Para esto se crea un archivo que contendrá los usuarios que deben ser previamente creados con htpasswd.

Archivo de grupo, por ejemplo el grupo **gente** contiene los usuarios **solange** y **adrian**

```
cat /usr/local/apache/passwd/grupo
gente: adrian solange
```

Normas de acceso para la autentificación

Se pueden almacenar en el archivo **httpd.conf** del servidor o en archivos .**htaccess** situados en los directorios donde se intenta acceder.

Veamos los pasos para restringir el acceso desde el archivo de configuración principal al subdirectorio secret situado en htdocs.

```
<Directory /usr/local/apache/htdocs/secret>
AllowOverride None
```

Impide modificaciones mediante archivos .**htaccess**

A partir de aquí todas las órdenes son IMPRESCINDIBLES

```
AuthType Basic [/Digest]
```

Tipo de autentificación: **básica** o **digest**. Distingue cómo se transmitirá la contraseña a través de la red, utilizando unincode (cifrado trivial) en el caso Basic, o utilizando MD5 (más seguro). La segunda posibilidad requiere compilar el apache con un módulo extra (**mod_digest.c**) y generar el archivo de contraseñas con **htdigest** en lugar de con **htpasswd**.

```
AuthUserFile /usr/local/apache/bin/passweb
```

Indica el nombre y la ruta del archivo de contraseñas. Debe especificarse la ruta completa, y no rutas relativas.

```
<Limit GET>
     require user pepito
</Limit>
</Directory>
```

La opción **Limit**: establece el tipo de acceso permitido (**GET**, **PUT** y **POST**) y para que usuario, se pueden especificar grupos.

Ejemplo de utilización de .htacces

Configuración en http.conf

```
AccessFileName .htaccess
<Directory /var/www/html/dir>
Order allow,deny
allow from all
AllowOverride All
</Directory>
```

Ejemplo configurando en el mismo `http.conf`
El usuario debe estar creado en el archivo `file.usuarios` con el comando `htpasswd`

```
<Directory "/var/www/html/personal">
    Options Indexes
    Order allow,deny
    Allow from all
        AuthType Basic
        AuthName "Restricted Files"
        AuthUserFile /usr/local/apache/passwd/file.usuarios
        require user solange
        <Limit GET POST OPTIONS>
                Order allow,deny
                Allow from all
        </Limit>
</Directory>
```

Ejemplo configuración completa para el uso de los directorios de los usuarios. Habilitando a los usuarios definidos en el grupo.

```
<IfModule mod_userdir.c>
    UserDir "disabled"
    UserDir "enabled" solange
    UserDir /home/*/www
</IfModule>

<Directory /home/*/www>
Order allow,deny
     allow from all
     AuthType Basic
     AuthName "Restricted Files"
     AuthUserFile /usr/local/apache/passwd/passwords
     AuthGroupFile /usr/local/apache/passwd/file.grupo
     <Limit GET>
     Order allow,deny
            allow from all
         require group usuarios
     </Limit>
</Directory>
```

Habilitar usuarios en general, definidos en el archivo.

```
<Directory "/var/www/html/privado">
    AuthName "Sitio privado -requiere autorización -"
    AUTHTYPE  BASIC
```

```
    AuthUserFile /usr/apache/password/users.apache
    require valid-user
    Options Indexes
</Directory>
```

Logs

HostNameLookups

```
HostNameLookups Off
```
Mantiene en el archivo de log el nombre del cliente si esta en on.

ErrorLog

```
ErrorLog "/var/log/httpd/error_log"
```

Localización del archivo para los log de errores.

LogLevel

```
LogLevel warn
```
Nivel del log, valores posibles: debug, info, notice, warn, error, crit, alert, emerg.

LogFormat

Formato de log personalizados:
```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Referencia de los valores de las variables en:
`manual/mod/mod_log_config.html#formats`

CustomLog

```
CustomLog logs/access_log common
```

Este log se utiliza para anotar las demandas del servidor, el formato será igual al definido por `common` en `LogFormat`. en el archivo logs/access_log