Tema 2: Administración de servidores web

Apache 2 y Apache 2.4

El servidor Web Apache sufrió un cambio muy profundo con la versión 2. Introdujo aspectos destacados como el uso de hebras y/o procesos en sistemas UNIX, y la inclusión de algunos módulos muy importantes; varios de estos módulos los trataremos en el punto correspondiente. Todos los cambios se pueden ver aquí.

La siguiente versión importante fue la 2.2 que en la actualidad sigue usándose mucho y es la versión que se ha utilizado en estos apuntes hasta esta versión en la que he decidido pasar a la 2.4, la más moderna.

Aunque los cambios más fuertes se producen en el paso de las versiones 1 a la 2, en las últimas versiones se han introducido cambios y mejoras que afectan a configuraciones y módulos que usaremos.

Otro punto a tener en cuenta es cómo actualizar de la versión 2.2 a la 2.4 que además servirá a aquellos familiarizados con la versión anterior de estos apuntes a familiarizarse con los nuevos usos.

Configuración del servidor web

En Apache, la configuración por defecto está en un archivo que se llama *httpd.conf*. Al instalar Apache se crea un directorio conf donde tenemos más archivos de ejemplo de configuración. Hasta el nombre y la ruta del archivo de configuración en Apache se pueden cambiar. En una instalación así estaría en

```
/etc/apache2/httpd.conf
```

Pero nada de esto se aplica a nuestra instalación ya que en las distribuciones de Ubuntu (y muchas otras precompiladas) el archivo base se llama *apache2.conf*

Antes de hacer ninguna modificación sobre este archivo debemos hacer una copia de seguridad del mismo (aquí vemos el del ejemplo pero es aplicable a cualquier archivo de configuración).

```
cd /etc/apache2/
sudo cp apache2.conf apache2.conf.old
```

En el archivo de configuración se incluyen directivas para el servidor y comentarios. Los comentarios son las líneas que comienzan por #. El servidor ignora todos los comentarios y las líneas en blanco así que podemos usar ambos para mejorar la legibilidad del archivo para el administrador.

Lo mejor es echar un vistazo al archivo de configuración; en nuestra distribución, el archivo principal de configuración es *apache2.conf*

```
gedit apache2.conf
```

El archivo contiene directivas para el servidor en sí mismo: dónde estará el directorio que contendrá los documentos web (por ejemplo archivos HTML), qué módulos se cargarán, etc. También puede contener información sobre hosts virtuales, pero eso lo veremos un poco más adelante.

Lo importante ahora es ver qué ha pasado con la configuración. Si vamos al final del archivo veremos unas directivas *Include* que apuntan a otros ficheros o directorios. Esta distribución de Apache 2 utiliza una división de la configuración en diferentes archivos para mejorar la organización.

> Investiga y explica el uso predeterminado de cada uno de los archivos de configuración que aparecen en *apache2.conf*, incluyendo éste mismo. No hace falta que aprendas el uso de cada uno, solo que puedas definir en una línea o dos el propósito de cada archivo.

Realmente todo esto nos lo explican al principio del archivo en los comentarios, en los que además nos damos cuenta de que lo que hemos denominado la distribución de Ubuntu realmente toma la forma de hacer las cosas de Debian.

```
#
# Summary of how the Apache 2 configuration works in Debian:
# The Apache 2 web server configuration in Debian is quite different to
# upstream's suggested way to configure the web server. This is because Debian's
# default Apache2 installation attempts to make adding and removing modules,
# virtual hosts, and extra configuration directives as flexible as possible, in
# order to make automating the changes and administering the server as easy as
# possible.
```

```
# It is split into several files forming the configuration hierarchy outlined
# below, all located in the /etc/apache2/ directory:
#
#   /etc/apache2/
#   |-- apache2.conf
#   |     `--  ports.conf
#   |-- mods-enabled
#   |     |-- *.load
#   |     `-- *.conf
#   |-- conf-enabled
#   |     `-- *.conf
#   `-- sites-enabled
#         `-- *.conf
#
#
# * apache2.conf is the main configuration file (this file). It puts the pieces
#   together by including all remaining configuration files when starting up the
#   web server.
#
# * ports.conf is always included from the main configuration file. It is
#   supposed to determine listening ports for incoming connections which can be
#   customized anytime.
#
# * Configuration files in the mods-enabled/, conf-enabled/ and sites-enabled/
#   directories contain particular configuration snippets which manage modules,
#   global configuration fragments, or virtual host configurations,
#   respectively.
#
#   They are activated by symlinking available configuration files from their
#   respective *-available/ counterparts. These should be managed by using our
#   helpers a2enmod/a2dismod, a2ensite/a2dissite and a2enconf/a2disconf. See
```

```
#   their respective man pages for detailed information.
#
# * The binary is called apache2. Due to the use of environment variables, in
#   the default configuration, apache2 needs to be started/stopped with
#   /etc/init.d/apache2 or apache2ctl. Calling /usr/bin/apache2 directly will not
#   work with the default configuration.
```

Un gran cambio que se produjo en Apache 2 es la introducción de los **módulos multiproceso** (Multiprocessing modules o MPMs). Hasta la versión 1.3 Apache funcionaba con un sistema "prefork" en el que un proceso creaba procesos hijo para atender las peticiones y él simplemente se encargaba de monitorizarlos para crearlos o destruirlos según fuera necesario en cada momento. Este sistema no funcionaba bien en determinados sistemas operativos (por ejemplo Windows) así que a partir de la versión 2 se usó otra solución:

Cada módulo MPM crea hilos (threads) o procesos hijo (mediante prefork) para atender las peticiones. Realmente hay varios módulos e incluso dependen del sistema operativo usado. Dependiendo de cuál usemos se usarán unas configuraciones u otras.

Una forma de saber qué módulo está usando Apache es usar la opción –V

En el servidor con Apache únicamente

```
apachectl –V
```

o

```
apache2 -V
```

nos muestra

```
Server version: Apache/2.4.7 (Ubuntu)
Server built:   Jul 22 2014 14:36:38
Server's Module Magic Number: 20120211:27
Server loaded:  APR 1.5.1-dev, APR-UTIL 1.5.3
Compiled using: APR 1.5.1-dev, APR-UTIL 1.5.3
Architecture:   64-bit
Server MPM:     event
  threaded:     yes (fixed thread count)
    forked:     yes (variable process count)
Server compiled with....
 -D APR_HAS_SENDFILE
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_SYSVSEM_SERIALIZE
 -D APR_USE_PTHREAD_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D DYNAMIC_MODULE_LIMIT=256
 -D HTTPD_ROOT="/etc/apache2"
 -D SUEXEC_BIN="/usr/lib/apache2/suexec"
 -D DEFAULT_PIDLOG="/var/run/apache2.pid"
 -D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D AP_TYPES_CONFIG_FILE="mime.types"
 -D SERVER_CONFIG_FILE="apache2.conf"
```

Podemos ver que usa event.

Vamos a echar un vistazo al archivo *apache2.conf* e intentar ver para qué servirá cada directiva que aparece.

Dividiendo y organizando el archivo de configuración

Lo primero que podemos observar es que *httpd.conf* no existe (hasta la versión 2.2 aparecía pero vacío).

```
sudo gedit /etc/apache2/httpd.conf
```

Como ya he comentado en una instalación manual, la configuración por defecto irá toda en ese archivo, pero viendo lo que sucede en la instalación que estamos utilizando, podemos aprender a organizar la configuración para hacerla más manejable.

Esto es porque el archivo principal de configuración puede cambiarse cuando compilamos Apache. Por defecto es *httpd.conf* pero en la instalación LAMP que usamos lo han cambiado a *apache2.conf* y han dejado el otro por si queremos hacer modificaciones. De cualquier forma podemos editar el archivo principal.

```
sudo gedit /etc/apache2/apache2.conf
```

A lo largo del archivo aparece una sección de inclusiones que permiten subdividir la configuración del servidor Apache como mejor nos parezca. En este caso se muestran:

```
# Include module configuration:
IncludeOptional mods-enabled/*.load
```

```
    IncludeOptional mods-enabled/*.conf

    # Include list of ports to listen on
    Include ports.conf

    # Include of directories ignores editors' and dpkg's backup files,
    # see README.Debian for details.

    # Include generic snippets of statements
    IncludeOptional conf-enabled/*.conf

    # Include the virtual host configurations:
    IncludeOptional sites-enabled/*.conf
```

La mayoría de ellos iremos viendo para qué sirven en sucesivas secciones. Baste aclarar que por ejemplo en *ports.conf* va información que ya conocemos y que esto puede hacerse con todo. Si nuestro servidor es pequeño quizá nos baste con usar un único fichero de configuración con todas las directivas pero en cuanto vaya a ser un poco complejo nos irá mejor si lo dividimos siguiendo algún criterio lógico.

Una curiosidad es que en varios *include* se añade un directorio. Suele usarse para crear en él archivos de configuración de elementos que no tienen otra sección lógica o para pruebas en la configuración y que así sea fácil revertir la situación al estado anterior en caso de fallo o para añadir configuración que tenga que ver con módulos o sitios concretos que pueden habilitarse o deshabilitarse según se necesite. Cuando instalemos Apache para el uso de módulos hablaremos de todos los archivos de configuración en más detalle.

NOTA: aunque no tiene que ver son la asignatura añado aquí cómo buscar texto dentro de archivos de un directorio y sus subdirectorios en Linux ya que con las diferentes subdivisiones de configuración y los cambios que se producen en ellos según la distribución y quién realice la configuración es muy útil.

```
    grep -iR textoabuscar|more
```

Directivas del archivo de configuración

Como ya hemos comentado antes, el archivo de configuración está compuesto por directivas que le indican al servidor cómo actuar. Ten en cuenta que cada vez que haces un cambio en el archivo de configuración es necesario reiniciar el servidor para que surta efecto. No todas las directrices que vamos a ver están en nuestro archivo de configuración. También existen otras y sobre algunas hablaremos más adelante.

En esta dirección está la referencia de las directivas de apache. Por último aquí podemos obtener una pequeña explicación y forma de uso de ellas con una indicación de para qué sirven. En esta última página en la tercera columna se indican los contextos en los que puede aplicarse cada directiva. Aquí se concreta un poco más a qué hace referencia cada contexto.

```
sudo apachectl restart
```

Es importante destacar que las líneas que comienzan por # **son comentarios o están comentadas**.

ServerRoot

Ahora vamos a comentar formalmente las directivas que aparecen en el archivo de configuración.

```
#ServerRoot "/etc/apache2"
```

Que indica el directorio raíz de la instalación de Apache. No se refiere al directorio donde colocaremos las páginas web. Ten en cuenta que está comentada porque es la que usa por defecto.

Conviene recordar que al realizar la instalación manual, antes de compilar el servidor lo configuraríamos con la siguiente línea, y estaríamos indicando que es /www

```
./configure --prefix=/www --enable-shared=max --enable-wmodule=rewrite --enable-module=so
```

Si nos fijamos, el directorio coincide con el valor de *--prefix*.

Esta directiva solo se modificaría en caso de mover el servidor Apache a otra ubicación en la estructura de directorios y como iremos viendo habría que cambiar más. Lo mejor es elegir bien desde el principio dónde instalaremos Apache.

En el caso que estamos viendo, no aparece ninguna otra ruta, pero para configurar rutas relativas al directorio de instalación de Apache, una vez especificado con la directiva anterior, podríamos usar *%ServerRoot%* en lugar de la ruta completa.

PidFile

PidFile establece la ruta al archivo en el que el servidor graba su ID de proceso (pid). Por defecto, el PID se coloca en *%ServerRoot%/logs*

En el archivo vemos que aparecen unas **constantes** que están definidas en el archivo */etc/apache2/envvars*

Para ver los valores reales puedes editar el archivo, nosotros usaremos ejemplos concretos porque son más claros.

```
PidFile logs/httpd.pid
```

No se recomienda cambiar la ruta si no se sabe muy bien lo que se está haciendo.

Timeout

Son los segundos que se esperan las respuestas durante la comunicación. Por defecto es 300 segundos y se recomienda no cambiarlo.

```
Timeout 300
```

KeepAlive, MaxKeepAliveRequests y KeepAliveTimeout

Determina si el servidor va a permitir que cada conexión haga más de una petición. El problema de activarlo es que un único cliente puede consumir demasiados recursos y saturar el servidor por lo que en caso de establecerlo a *on* se recomienda configurar cuidadosamente *KeepAliveTimeout*, generalmente a un nivel bajo (en la versión 2.2 por defecto era 15 y se ha bajado a 5).

```
KeepAlive Off
```

MaxKeepAliveRequests determina el número de peticiones que podrá realizar cada conexión. Evidentemente solo tiene sentido si *KeepAlive* está activada.

```
MaxKeepAliveRequests 100
```

KeepAliveTimeout determina el tiempo que el servidor esperará antes de atender una nueva petición del mismo cliente en la misma conexión.

```
KeepAliveTimeout 5
```

Estas directivas son un buen ejemplo de elementos que pueden hacer que nuestro servidor no funcione como esperamos una vez en producción. Si no probamos con un número de peticiones superior al máximo no podremos comprobar si el funcionamiento es el esperado.

IfModule

Es un contenedor que permite establecer determinadas opciones solo si se ha cargado un módulo determinado. Si se escribe ! (cierre de exclamación) antes del nombre del módulo se ejecutan las opciones si no se ha cargado el módulo.

Esta directiva no aparece en la configuración principal porque se ha desplazado al archivo de configuración de cada módulo, pero se mantiene aquí porque su uso es importante.

```
<IfModule mod_mime_magic.c>
    MIMEMagicFile conf/magic
</IfModule>
```

StartServers

Esta directiva aparece en el archivo */etc/apache2/mods-enabled/mpm_event.conf*

```
<IfModule mpm_event_module>
    StartServers              2
    MinSpareThreads           25
    MaxSpareThreads           75
    ThreadLimit               64
    ThreadsPerChild           25
    MaxRequestWorkers         150
    MaxConnectionsPerChild    0
</IfModule>
```

Apache crea y destruye servidores automáticamente según el tráfico que tenga que atender en cada momento. Apache es muy eficiente en esto por lo que no deberíamos preocuparnos demasiado de este y de los siguientes parámetros. Además deberían ir dentro de un *IfModule* según al que queramos aplicarlo.

Esta directriz establece cuantos servidores se crearán al arrancar.

```
StartServers 5
```

Al final de la explicación de la directiva podemos ver que se usan otras asociadas a esta para determinar cuántos servidores o hilos (hebras) se deben mantener a la espera.

Listen

Esta directiva se encuentra en *ports.conf*

Este elemento indica al servidor en qué dirección y puerto debe escuchar las peticiones http que lleguen **además de los de por defecto**. El puerto estándar que un servidor web reciba peticiones http es el 80 por lo que la línea que nos encontramos es

```
Listen 80
```

En nuestro caso hemos instalado Apache en una máquina virtual así que está configurado para escuchar en la interfaz de loopback (127.0.0.1). Si queremos que el servidor sea visto desde la máquina host debemos configurarlo situando la dirección IP que configuramos para la tarjeta de red que añadimos como solo-anfitrión. La dirección IP y el puerto se separan por dos puntos.

```
Listen 192.168.56.101:80
```

Si configuramos Apache para escuchar en otro puerto solo podremos acceder a las páginas web añadiendo dicho puerto detrás de la dirección. Como vimos al instalar Tomcat, éste escucha por defecto en el puerto 8080 y por ello accedíamos a él con http://localhost:8080

LoadModule

Cuando instalamos Apache, habilitamos la opción de Dynamic Shared Object (DSO) que permite añadir módulos dinámicamente sin necesidad de recompilar el servidor. La directiva LoadModule indica qué módulos dinámicos cargar. No es necesario incluir los módulos que se compilaron con el servidor.

En el archivo aparece comentada porque no añadimos ningún módulo. Más adelante hablaremos de los módulos.

De todas formas en nuestra distribución se ha optado por otro método que veremos más adelante.

User y Group

Estas dos directivas indican con qué usuario y grupo se lanzarán los procesos hijos que genere Apache. Estos procesos hijo no deben lanzarse con el usuario *root* por razones de seguridad ya que crearían una brecha perfecta para los hackers.

Si no arrancamos el servidor con el usuario *root*, los procesos hijo que se lanzarán con ese mismo usuario ya que solo *root* puede cambiar el usuario y el grupo de un proceso por lo que estas directivas se ignorarán.

En Linux por defecto el usuario *nobody* y el grupo *nogroup* tienen muy pocos privilegios por lo que son buenos candidatos para lanzar los procesos hijo.

Si queremos usar el número del grupo o del usuario en lugar del nombre debemos añadir # justo antes del número.

```
User nobody
Group #-1
```

En nuestra distribución se utilizan dos variables globales.

ServerAdmin

Esta directiva y las siguientes aparecen en */etc/apache2/sites-available/000-default.conf*

Esta opción permite configurar la dirección del administrador del servidor web que se mostrará si el servidor genera una página de error. Evidentemente debe ser una dirección real y generalmente es del mismo domino que el propio servidor web. Nosotros no disponemos de un nombre de dominio, pero podemos configurar una dirección de correo real.

```
ServerAdmin profesor.scv@gmail.com
```

ServerName

Indica el nombre del servidor. Debe cumplir con las especificaciones DNS y estar en nuestro poder.

```
ServerName www.ejemplo.es:80
```

DocumentRoot

Con esta opción indicamos el directorio raíz donde colocaremos las páginas web. Podemos crear subdirectorios dentro de éste y accederemos a los documentos que pongamos en el subdirectorio con una ruta relativa.

```
DocumentRoot "/www/docweb"
```

El problema es que no basta con cambiar esta ruta, es necesario cambiar también otra de la directiva Directory que veremos luego.

```
<Directory "/www/docweb">
…
```

Si queremos comprobar que funciona lo mejor es modificar el archivo index.html del nuevo directorio o añadir un archivo nuevo y cargar ese.

En el caso que nos ocupa ahora, esta directiva aparece en el archivo *default* del directorio */etc/apache2/sites-available*. Esto es así porque la instalación que usamos nosotros viene prepearada por defecto para funcionar con sitios virtuales.

Directory

Esta directiva había sido derivada al archivo de configuración del sitio por defecto, para que no se aplicara a todos los sitios del servidor ya que no permite sobrescritura de las directivas de seguridad. En nuestro caso está en el servidor por lo que afectaría a todos los sitio que utilizaran el directorio especificado.

Esta opción se usa para configurar cómo se comportará y qué se permitirá en cada directorio al que tiene acceso el servidor Apache. Esta configuración se aplica a un directorio y los subdirectorios que contiene si no se sobrescribe en otra definición sobre un directorio más concreto.

Nos encontramos dos veces esta etiqueta. La primera hace referencia al directorio raíz y se configura siempre con opciones muy restrictivas.

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>
```

que impide que nadie que accede al servidor pueda acceder a ningún directorio de la estructura de nuestro servidor ya que se aplica a los subdirectorios también y estamos partiendo del directorio raíz.

Esto es una de las cosas que ha cambiado desde la versión 2.2 ya que antes se utilizaba una sintaxis diferente a la hora de establecer los permisos.

```
Notación 2.2
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
```

```
        Deny from all
    </Directory>
```

Hay que tener mucho cuidado porque en el momento de la realización de estos apuntes, en la documentación de *directory* el ejemplo que lo usa utiliza la notación antigua sin embargo en la página para actualizar de la versión 2.2 a la 2.4 está bien.

El contenido lo vemos a continuación. Primero vamos a mostrar también la otra vez que aparece, haciendo referencia al directorio *DocumentRoot* y al compartido del usuario.

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

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

Aunque ambos casos estén configurados igual, podrían diferir.

Pueden añadirse directivas para otros directorios según lo vayamos necesitando.

La primera línea permite a Apache seguir enlaces simbólicos y la segunda que las opciones de acceso de cada directorio (archivo *.htaccess* del que hablaremos en la directiva *AccessFileName*) no priman sobre éstas. La tercera directiva indica los permisos. Por ahora nos basta saber que se permite el acceso a todos (y en la anterior se denegaba a todos).

DirectoryIndex

Esta directiva en la actualidad se encuentra en */etc/apache2/mods-enabled/dir.conf*

Especifica la página por defecto que se buscará al acceder a un directorio de la jerarquía de nuestro sitio. Acceder a un directorio es usar una dirección web que acaba con una barra "/". El directorio raíz está incluido en esa jerarquía por lo que cuando accedemos a nuestro sitio web (en nuestros casos http://localhost) carga el archivo index.html que es el valor por defecto de *DirectoryIndex*.

```
    DirectoryIndex index.html
```

Puede establecerse una sucesión de archivos y el servidor mostrará la primera que encuentre del orden establecido en la directiva.

```
    DirectoryIndex index.html, index.htm, inicio.html, inicio.htm
```

Si accedemos a un directorio que no contiene ninguno de los archivos especificados, Apache crea dinámicamente un archivo que lista los contenidos.

AccessFileName

Indica el nombre del archivo en el que se deben buscar las directivas de acceso determinadas en cada directorio. Por defecto es *.htaccess* y no se recomienda cambiarlo en absoluto.

```
    AccessFileName .htaccess
```

Files

Es un contenedor que sirve para establecer directivas para tipos de archivo. Por lo menos es necesario añadir un grupo que impida el acceso a los archivos que empiezan por .ht por motivos de seguridad.

```
<FilesMatch "^\.ht">
    Require all denied
</FilesMatch>
```

Se puede usar para otros tipos de archivo según creamos conveniente.

HostnameLookups

Indica al servidor si debe hacer una consulta DNS para cada petición. Esto consume mucho tiempo por lo que por defecto está deshabilitada.

```
HostnameLookups off
```

ServerSignature

Esta directiva se encuentra en el archivo */etc/apache2/conf-available/security.conf*

Indica si al mostrarse una página generada automáticamente por el servidor (no las generadas mediante lenguajes a partir de algo establecido por el usuario sino las páginas de error, listado de directorios FTP, etc) debe mostrarse el nombre y la versión del servidor. Esto puede ser usado maliciosamente así que si no estás conforme establécelo a off.

```
ServerSignature On
```

Hay otra opción (EMail) que añade la dirección del administrador de la web a la información mostrada.

Alias

Esta directiva se encuentra en el archivo */etc/apache2/mods-enabled/alias.conf*

Permite crear alias para archivos o directorios. Se pueden añadir los que queramos pero el más habitual es el de la carpeta de iconos que usa Apache.

```
Alias /icons/ "/usr/share/apache2/icons/"

<Directory "/usr/share/apache2/icons">
    Options FollowSymlinks
    AllowOverride None
    Require all granted
</Directory>
```

La directiva es solo la primera línea. Lo demás es la directiva de configuración del directorio.

ScriptAlias

Se encuentra en */etc/apache2/conf-enabled/serve-cgi-bin.conf*

Indica dónde se ubicará la carpeta de scripts CGI. El funcionamiento es similar al anterior pero solo debemos incluirlos si vamos a usar scripts CGI.

```
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
    AllowOverride None
    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
    Require all granted
</Directory>
```

IndexOptions, AddIconByEncoding, AddIconByType, AddIcon, DefaultIcon, ReadmeName, HeaderName e IndexIgnore

Son directivas asociadas a la creación de índices de manera automática por el servidor. Cuando nos muestra el servidor web un listado de contenidos de un directorio por ejemplo. Están en el archivo */etc/apache2/mods-enabled/autoindex.conf*

No vamos a profundizar en ellas.

LanguagePriority

Tanto esta directiva como la siguiente tienen mucho que ver son el módulo MIME que veremos más adelante aunque no forman parte de él y con los conocimientos que traemos del primer curso podemos entender cómo funcionan.

Permite establecer una prioridad de los idiomas en caso de que no se especifique uno o haya un empate en la negociación por diferentes motivos. Por defecto viene en inglés, pero en la mayoría de los casos nosotros querremos establecerlo en español.

```
LanguagePriority es en fr de
```

Para poder probar este tipo de configuración es necesario tener varios archivos en diferentes idiomas con el mismo nombre. En ésta página podemos ver cómo hacerlo, pero aunque es bastante completa es antigua. En el W3c nos proporcionan una información más reciente y en la documentación oficial de Apache aparece todo especificado.

Los códigos de idiomas están mantenidos por la IANA. Como son los mismos que se utilizan en HTML, XML, etc existen páginas muy completas sobre ellos.

AddDefaultCharset

Debería establecerse al juego de caracteres que mejor se ajuste a la zona en la que se sitúa el servidor y al idioma del contenido. En caso de no estar seguros es mejor dejarlo como está.

```
AddDefaultCharset ISO-8859-1
```

BrowserMatch

Se encuentran (entre otros) en */etc/apache2/mods-enabled/setenvif.conf*

Sirve para modificar la respuesta dependiendo de la configuración del cliente en cuanto a navegador y plugins. Esto suele usarse para evitar problemas con navegadores que no siguen algún estándar o evitar agujeros de seguridad. Las siguientes líneas son muy comunes porque incluyen configuraciones con problemas de sobra conocidos.

```
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

Configura tu servidor Apache para que la carpeta raíz sea /pagweb y prueba a consultarlo desde la máquina host. Solo admitirá consultar por el puerto 9090. Además la página por defecto que cargará en cada directorio debe ser inicio.html. Por último añade el idioma español, catalán, gallego y vasco en este orden. El juego de caracteres por defecto debe ser el europeo occidental. Prueba el correcto funcionamiento de todo.

Registros de error

Los archivos de son fundamentales en la gestión de servidores ya que nos permiten comprobar qué ha sucedido en cada momento. Aunque hay otros tipos, los que veremos ahora son los que guardan errores de configuración o funcionamiento.

Estas directivas de registro de errores pueden ir tanto en la configuración principal como en la de algún sitio (host) específico. Aunque no nos extenderemos por falta de tiempo sí vamos a echar un vistazo general al registro, ya que es tremendamente útil cuando tenemos algún problema de funcionamiento en el servidor o alguno de los sitios web enlazados.

Lo primero es echar un vistazo general a la documentación de Apache para hacernos una idea.

ErrorLog

Esta directiva es muy importante ya que indica dónde ubicar el archivo de registro de los errores que se produzcan en el servidor. El lugar por defecto es *%ServerRoot%/logs/error_log*. Muchos administradores crean una partición exclusivamente para situar este tipo de archivos y así tener más probabilidades de poder consultarlos en caso de un error fatal.

```
    ErrorLog logs/error_log
```

LogLevel

Establece cuánta información se guardará en el archivo de registro de errores. El nivel por defecto es suficiente para empezar pero cuanto mayor sea el servidor y más importante su función más información necesitaremos. Los valores posibles incluyen: debug, info, notice, warn, error, crit, alert, emerg.

```
    LogLevel warn
```

LogFormat
Establece qué y en qué formato se registrará. No vamos a entrar en más detalles en este curso.

CustomLog
Establece la ruta al archivo de que registra las visitas a nuestro sitio web.

```
#CustomLog logs/access_log combined
```

Combined indica el uso de un único archivo para guardar toda la información ya que es posible dividir este registro en varios.

> Consulta los archivos de registro de errores y de acceso de tu servidor Apache.

Módulos

El diseño de Apache es modular, algo que ya hemos dejado entrever cuando hemos hablado de Dynamic Shared Object (DSO). El núcleo de Apache incluye la funcionalidad necesaria para establecer un servidor web pero existen muchos módulos adicionales que permiten añadir funciones extra.

El uso de DSO tiene unas ventajas evidentes y la única desventaja es una pequeña disminución del rendimiento del servidor por lo que se recomienda su utilización excepto en casos en los que el rendimiento sea crítico.

Cada módulo tiene un conjunto de directivas específicas que permiten su gestión. Nosotros ahora nos centraremos en ver qué módulos hay y más adelante estudiaremos algunos casos en profundidad.

Puedes consultar los módulos que están instalados en cada servidor mediante el comando

```
sudo apachectl –l
```

que en el caso de nuestra instalación nos devuelve lo siguiente.

```
Compiled in modules:
  core.c
  mod_so.c
  mod_watchdog.c
  http_core.c
  mod_log_config.c
  mod_logio.c
```

```
      mod_version.c
      mod_unixd.c
```

Otra forma de consultarlo es

```
sudo apachectl –M
```

Que nos muestra los módulos por nombre y determinando si se añadieron de forma estática (cuando se compiló Apache) o de manera dinámica.

```
Loaded Modules:
 core_module (static)
 so_module (static)
 watchdog_module (static)
 http_module (static)
 log_config_module (static)
 logio_module (static)
 version_module (static)
 unixd_module (static)
 access_compat_module (shared)
 alias_module (shared)
 auth_basic_module (shared)
 authn_core_module (shared)
 authn_file_module (shared)
 authz_core_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 autoindex_module (shared)
```

```
     deflate_module (shared)
     dir_module (shared)
     env_module (shared)
     filter_module (shared)
     mime_module (shared)
     mpm_event_module (shared)
     negotiation_module (shared)
     setenvif_module (shared)
     status_module (shared)
```

Es fácil reconocer el mismo módulo en la terminología del primer listado y en la del segundo.

Puedes consultar todos los módulos aquí. Si pinchas en el nombre de alguno te llevará a una explicación más completa y con ejemplos.

Vamos a presentar algunos de los diferentes módulos en grupos con funciones similares.

> Para los siguientes módulos debes leer la pequeña explicación que viene aquí, prepararte para explicar qué hace y pensar uno o dos ejemplos de utilidad que tenga el módulo. Vamos a irlo haciendo grupo a grupo.

La versión 2.4 trae módulos nuevos y mejoras en otros ya existentes.

El módulo más importante es el denominado Core (corazón o núcleo) que contiene la mayoría de las directivas vistas hasta ahora y muchas otras.

Módulos relacionados con el entorno

Estos módulos nos permiten controlar qué parte del entorno del servidor estará disponible para otros módulos o programas. Al decir entorno nos referimos al conjunto de variables dinámicas de entorno de Apache que modifican el comportamiento del servidor. Pueden ser útiles en otros módulos (aquí me refiero a “asignaturas”) como Desarrollo en Entorno Servidor.

mod_env

Este módulo está disponible en nuestras instalaciones. Permite pasar el valor de variables de entorno a programas de script CGI, perl, PHP, etc.

mod_setenvif

También aparece por defecto por lo que está disponible en nuestras instalaciones. Este módulo nos posibilita la creación de variables de entorno a partir de datos que nos envía en cliente con el protocolo HTTP. La directiva *BrowserMatch* vista anteriormente pertenece a este módulo.

mod_unique_id

Este módulo sin directivas se encarga de establecer un identificador único para cada petición que llega a la máquina con el servidor Apache en caso de que lo necesitemos. Realmente puede crear un identificador único para cada petición que llegue a un conjunto de máquinas configuradas correctamente para funcionar como un cluster.

Módulos de autenticación y control de acceso

Apache implementa varios módulos para realizar autenticación y control de acceso lo que se suele aplicar al filtrado de usuarios que pueden visitar un directorio de nuestro web basándose en la dirección IP o el nombre de usuario.

El mayor problema que presentan estos módulos es que utilizan el protocolo HTTP que transmite texto plano por lo que las contraseñas se envían sin cifrar creando un agujero de seguridad muy preocupante.

Hablaremos más de varios de estos módulos cuando veamos autenticación y autorización.

mod_authXXXXXX

mod_auth_basic es el módulo básico de autenticación de Apache. El módulo mod_auth_digest es similar pero permite encriptar la contraseña en lugar de enviarla de de forma "clara". Usan el protocolo HTTP para ello.

Relacionados con él hay varios módulos más con diferentes formas de autenticación (mod_authn_xxxx) y autorización (mod_authz_xxxx) de los que los ejemplos más sencillos son mod_authn_file y mod_authz_user.

En Apache 2.4 se ha incluido mod_auth_form.

> En parejas o por grupos, intentad entender cómo funcionan estos módulos de manera básica (no se pretende que podamos hacerlos funcionar aún) y de qué se encarga cada uno de los tres grupos. Posteriormente lo pondremos en común.
>
> Una vez hemos entendido los grupos vamos a ver el uso de uno u otro dentro de cada grupo, prestando especial atención a los que vienen por defecto en Apache.
>
> ¿Crees que en versiones anteriores de Apache no podía usarse la autenticación mediante formularios? ¿qué aporta la inclusión del módulo mod_auth_form?

mod_authz_host

Permite autenticación mediante dirección IP o nombre de dominio.

mod_access permitía en versiones anteriores activar la autenticación mediante dirección IP o nombre de la máquina cliente. Sin embargo usa la sintaxis antigua por lo que está desaconsejado.

Módulos de generación dinámica de contenidos

Estos módulos permiten delegar la atención de determinadas peticiones a diferentes scripts o programas externos.

mod_cgi

Sirve para ejecutar scripts de tipo CGI (Common Gateway Interface)

mod_include

Permite usar filtros SSI (Server-Side Includes)

mod_actions

Dependiendo del tipo MIME o en el método de la petición HTTP, permite usar diferentes scripts para procesar dichas peticiones.

mod_ext_filter

Permite filtrar una respuesta mediante un programa externo antes de enviársela al cliente.

Módulos de configuración del tipo de contenido

Este conjunto de módulos permiten al servidor detectar o negociar el tipo de contenido más adecuado para el cliente, entendiendo como cliente la máquina y el navegador que recibirán la respuesta HTTP.

mod_mime

Permite que Apache determine el tipo MIME a partir de la extensión del archivo. Se compila por defecto y nos permite activar un *Handler* para gestionar tipos de archivo. Varias de las directivas que vimos de tipos de codificación y de idiomas del contenido están relacionadas con este módulo.

mod_mime_magic

Este módulo permite a Apache determinar el tipo MIME a partir de un patrón de bytes que se almacena en un archivo y se compara con la petición. Sólo se activa si el módulo anterior no es capaz de determinar el tipo MIME.

mod_negotiation

La negociación de contenido entre cliente y servidor típica consiste en que el cliente le envía al servidor qué tipos de contenido (idioma, codificación, etc) puede manejar y el servidor busca el más adecuado para responderle. Este módulo se compila por defecto.

Módulos para el listado de directorios

Cuando un cliente hace una petición a nuestro servidor con una dirección web que indica un directorio el servidor intenta servir primero el archivo o archivos establecidos para ello (con la directiva *DirectoryIndex*). En caso de no encontrar ninguno en ese directorio muestra una lista de los archivos contenidos. Estos módulos nos permiten configurar cómo serán esos listados.

mod_dir

Módulo básico de manejo de directorios que se incluye en la compilación estándar de Apache. Por defecto realiza dos funciones básicas: añade una barra "/" al final de cada dirección que no termina en un nombre de archivo (redirige www.misitio.es/undirectorio a www.misitio.es/undirectorio/) y busca un archivo por defecto para cargar en los casos en los que se intenta acceder a un directorio. Si no definimos nada con la directiva *DirectoryIndex* se busca *index.html* por defecto.

mod_autoindex

Este modulo también se incluye en la configuración de Apache por defecto. En caso de que al acceder a un directorio no se encuentre ninguno de los archivos especificados *mod_autoindex* se encarga de generar el listado. Además podemos configurar cómo se generará dicho listado.

Configurar este módulo es útil en caso de que tengamos un servidor de archivos (por ejemplo en una intranet) al que accedan muchos usuarios. No hay que confundir este tipo de acceso con el que proporciona un servidor FTP como veremos en el tema 4.

Módulos para la gestión de las cabeceras HTTP de las respuestas

Las cabeceras en el protocolo HTTP incluyen mucha información importante para la comunicación. Éste conjunto de módulos nos permite modificar dichas cabeceras. El uso de estos módulos requiere conocimientos avanzados del funcionamiento del protocolo HTTP por lo que solo los listaremos: *mod_asis, mod_headers, mod_expires* y *mod_cern_meta* son los módulos que se incluyen en este grupo. Solo el primero se incluye en la compilación por defecto.

Módulos de información del servidor y de registro de la actividad

Estos módulos proporcionan información sobre el estado del servidor y permiten configurar el registro de la actividad.

mod_log_config

Permite configurar el registro del acceso de usuarios al servidor.

mod_status

Muestra información sobre el estado del servidor.

mod_info

Muestra información de configuración del servidor.

mod_usertrack

Permite identificar usuarios y registrarlos de manera individual usando HTTP Cookies. El identificar usuarios de forma individual nos permite tratar a cada usuario de forma única o servirle información personalizada por ejemplo.

> ¿Qué son las cookies? Investiga en Internet y explica de forma básica cómo funcionan. Recientemente Google y Microsoft han manifestado su intención de dejar de usar cookies, ¿significa esto que están dispuestos a renunciar a la funcionalidad que proporcionan? Entérate en Internet de qué plantean.
>
> ¿Dónde se almacenan las cookies en tu ordenador? Accede a alguna para ver su contenido. La extensión Edit this Cookie the Google Chrome puede ayudarte si exportas alguna y la pegas en un documento de texto.

Módulos de mapeo de URLs

Con este conjunto de módulos podemos manejar y modificar las URLs de nuestro sitio: a partir del nombre de domino hacia adelante. Nos permitirá crear alias, nos ayudará a tener varios sitios web en el mismo servidor y podremos rescribir las direcciones para que lleven a diferentes lugares de la estructura de archivos y directorios de nuestro servidor.

mod_userdir

Ya comentamos al revisar el archive de configuración de Apache cómo podíamos crear sitios personales para cada usuario usando la directiva *Userdir*. Este módulo es el que nos permite hacerlo.

mod_alias

Si queremos establecer “alias” o enlaces simbólicos entre dos rutas de la estructura de archivos. Incluso permite crear redirecciones de un archivo o directorio a otro. Este módulo se compila por defecto con Apache.

mod_rewrite

Con este módulo podemos modificar la URL de la petición que hace el cliente para que sea una que configuremos nosotros. Para ello, se establece un patrón con el que se compara la URL y si coincide se cambia por otra según otro patrón que se establece. Esto permite entre otras muchas cosas modificar la estructura de archivos y directorios de nuestro navegador web y que las URLs de la estructura antigua sigan funcionando. Es más potente que los alias creados en el módulo anterior porque un patrón puede englobar muchas URLs (todas las que cumplan con el patrón establecido).

mod_speling

Este módulo corrige posibles pequeños errores en las URLs de peticiones por parte de los clientes. Tiene dos tipos de corrección: en la primera permite un error como la introducción de un carácter de más, la omisión de un carácter o el cambio de un carácter por otro (solo uno) mientras que la segunda funcionalidad busca errores provocados por el incorrecto uso de mayúsculas y minúsculas.

El módulo compara la petición con los directorios y archivos que encuentra en la estructura. Si hay una coincidencia se realiza una petición de redirección al cliente y si hay varias se le envía al cliente la lista de coincidencias.

El gran problema que tiene la activación de este módulo es el tremendo impacto negativo que puede llegar a tener en el rendimiento del servidor por lo que hay que estar muy seguros de la necesidad de activarlo.

mod_vhost_alias

Este módulo está relacionado con el uso de hosts virtuales que veremos más adelante. Básicamente sirven para tener varios sitios web en el mismo servidor Apache. Sin embargo este módulo no se usa muy habitualmente porque lo que permite es la creación de hosts virtuales de forma dinámica y solo se recomienda cuando se van a crear muchísimos y la configuración manual vaya a ser demasiado lenta. Por ejemplo si un proveedor de servicios de Internet (ISP) decide crear sitios web para todos los clientes que lo soliciten.

Otros módulos

En este grupo vamos a hablar de varios módulos que no encajan en ninguno de los grupos anteriores pero que merece la pena tener en cuenta.

mod_so

Este es el módulo que nos permite añadir otros sin la necesidad de recompilarlos. Es el módulo que nos permite el uso de DSO (Dynamic Shared Object) y todos los demás módulos de Apache se pueden usar de esta manera menos este. Recuerdo que cuando compilamos Apache añadimos este módulo para poder usar otros sin necesidad de recompilar el servidor cada vez que quisiéramos añadir o eliminar algún módulo de nuestro servidor web.

mod_imagemap

Con éste módulo que se compila por defecto con Apache, se incluye el soporte para mapas de imágenes el archivos HTML.

mod_proxy

Este módulo nos permite convertir Apache en un servidor proxy. Un servidor proxy se sitúa entre el cliente y el servidor y básicamente actúa como el servidor para el cliente y como un cliente para el servidor. Los motivos para usar un servidor proxy pueden ser muy variados, pero incluyen dar acceso a determinados recursos de Internet a ordenadores sin direcciones viables, hacer la función de caché para los usuarios de una red o control, registrar y/o restringir el uso de determinados recursos de Internet.

mod_file_cache

Si usamos este módulo permitimos que Apache ponga en caché determinados archivos estáticos y no cambien frecuentemente. El problema con el uso de esta técnica es que si el archivo cambia puede tardar bastante en enviarse el archivo modificado a los clientes debido a la copia que tenemos en caché.

mod_dav

Si queremos usar WebDAV tendremos que activar este módulo. Hablaremos de WebDAV en el tema 4.

mod_example_hooks

Es un módulo de vital importancia para todo aquel que quiera o necesita aprender a programar un módulo de Apache, pero para nadie más. Hay que tener conocimientos del lenguaje de programación C para poder comprenderlo.

Instalación, configuración y uso de los módulos de Apache

Después de la introducción a las funciones de algunos módulos de Apache, vamos a aprender a usar algunos de ellos. Aunque cada uno tiene sus propias peculiaridades la práctica con algunos nos llevará a comprender el funcionamiento y poder usar cualquiera consultando la documentación pertinente.

Los archivos de configuración de la instalación por paquete

Este tipo de instalación lleva asociada una distribución de las directivas de configuración en varios archivos. Es mucho más racional y simplifica la administración del servidor Apache con respecto a la instalación manual que utiliza un único fichero de configuración aunque mediante la directiva *Include* se pude configurar al gusto de cada uno como vimos en el apartado correspondiente.

Realmente la parte que nos afecta a a nosotros es

```
# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
```

Realmente en la carpeta *mods-enabled* solo hay enlaces símbólicos a los archivos en *mods-available*.

Comprobación de los módulos ya instalados

Para comprobar qué módulos están instalados por defecto

```
apachectl -M
```

Vemos lo siguiente

```
Loaded Modules:
 core_module (static)
 so_module (static)
 watchdog_module (static)
 http_module (static)
 log_config_module (static)
 logio_module (static)
 version_module (static)
 unixd_module (static)
 access_compat_module (shared)
 alias_module (shared)
 auth_basic_module (shared)
 authn_core_module (shared)
 authn_file_module (shared)
 authz_core_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 autoindex_module (shared)
```

```
 deflate_module (shared)
 dir_module (shared)
 env_module (shared)
 filter_module (shared)
 mime_module (shared)
 mpm_event_module (shared)
 negotiation_module (shared)
 setenvif_module (shared)
 status_module (shared)
```

Por lo que al llevar activo el *so_module* quiere decir que soporta DSO así que se pueden activar los módulos de manera dinámica.

Nosotros podemos encontrar los módulos en la carpeta */usr/lib/apache2/mods-available*

```
cd /usr/lib/apache2/mods-availabe
ls –la
```

Existen muchos módulos adicionales que no se incluyen en la instalación estándar de Apache. Para consultar estos módulos debemos ejecutar la orden

```
sudo apt-cache search libapache2-mod
```

Instalación

Para instalar un módulo de Apache hay que usar la directiva *LoadModule* del módulo mod_so. Cuya sintaxis viene definida como

```
LoadModule module filename
```

Donde *module* es el nombre del módulo y *filename* la ruta y nombre del archivo *.so* donde se encuentra.

Por ejemplo para activar el módulo *mod_speling*

```
LoadModule speling_module /usr/lib/apache2/modules/mod_speling.so
```

Siendo necesario después reiniciar Apache

```
apachectl restart
```

> ¿Qué sucede si intentas cargar un módulo que no existe?

Pero en nuestra instalación vemos que todo esto está organizado en diferentes archivos. Si vamos a la carpeta */etc/apache2/mods-available* veremos que hay dos extensiones de archivo:

xxxxx.load que contiene la instrucción para cargar el módulo.

```
LoadModule alias_module /usr/lib/apache2/modules/mod_alias.so
```

xxxxx.conf que contiene la configuración para dicho módulo.

```
<IfModule alias_module>
    # Aliases: Add here as many aliases as you need (with no limit). The format is
    # Alias fakename realname
    #
    # Note that if you include a trailing / on fakename then the server will
    # require it to be present in the URL.  So "/icons" isn't aliased in this
    # example, only "/icons/".  If the fakename is slash-terminated, then the
    # realname must also be slash terminated, and if the fakename omits the
    # trailing slash, the realname must also omit it.
    #
    # We include the /icons/ alias for FancyIndexed directory listings.  If
    # you do not use FancyIndexing, you may comment this out.

    Alias /icons/ "/usr/share/apache2/icons/"

    <Directory "/usr/share/apache2/icons">
        Options FollowSymlinks
        AllowOverride None
        Require all granted
    </Directory>

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Esta forma de funcionar simplifica todo pero podrían usarse igual desde un único archivo de configuración.

Alternativa

En algunas distribuciones de Linux hay otra forma de activar los módulos. Si nos damos cuenta tenemos dos directorios /etc/apache2/mods-available y /etc/apache2/mods-enabled. En el primero están todos los módulos que vienen incluidos en esta versión de Apache mientras que en el segundo están los que tenemos activos en nuestro servidor mediante enlaces simbólicos a los módulos en el directorio anterior.

Para activar un módulo usamos el comando a2enmod con el nombre del módulo.

```
sudo a2enmod speling
```

que nos informa de lo que está sucediendo y de que es necesario reiniciar el servidor

```
Enabling module speling.
To activate the new configuration, you need to run:
  service apache2 restart
```

y para desactivarlo se hace igual pero con el comando a2dismod.

```
sudo a2dismod speling
```

para lo que obtenemos una información equivalente.

```
Module speling disabled.
To activate the new configuration, you need to run:
  service apache2 restart
```

después de cada activación o desactivación es necesario reiniciar el servidor.

```
apachectl restart
```

ten en cuenta que la línea anterior es equivalente a

```
apache2 restart
```

Uso y configuración

Cada módulo en Apache 2 tiene un uso y configuración diferentes. Para saber cómo usar cada módulo lo mejor es consultar la documentación de la página oficial. Desde la lista de módulos se puede hacer click en cada uno para ver sus directivas y uso.

Nosotros vamos a seguir con el ejemplo del módulo *mod_speling*. Podemos ver que solo tiene dos directivas que podemos usar: *CheckSpelling* y *CheckCaseOnly*. La primera activa o desactiva el módulo y la segunda hace que solo se corrijan los errores de mayúsculas/minúsculas. Para activarlo la corrección completa debemos añadir al archivo de configuración la siguiente línea:

```
CheckSpelling on
```

Después de cada cambio en la configuración es necesario reiniciar la máquina para que la modificación surta efecto.

```
apache2 restart
```

Para probar si todo funciona correctamente podemos intentar cargar la página web http://localhost/insdex.html antes de los cambios en el servidor. Nos devuelve una página de error generada automáticamente con Apache con el siguiente texto:

```
Not Found

The requested URL /insdex.html was not found on this server.
Apache/2.2.22 (Ubuntu) Server at localhost Port 80
```

Si después de los cambios intentamos acceder a la misma página web veremos que corrige la dirección a http://localhost/index.html y carga la página de inicio de Apache.

Prueba a activar el módulo que corrige la sintaxis modificando el archivo httpd.conf (no el apache2.conf ya que así es más fácil comprobar lo que hemos hecho). Prueba que todo funciona correctamente y borra las modificaciones en la configuración.

Ten en cuenta la directiva IfModule

Devuelve la configuración al estado anterior a este ejercicio.

Archivos de configuración asociados a cada módulo

En el directorio */etc/apache2/mods-enabled* podemos ver que hay dos archivos asociados a cada módulo activo. Los archivos .load incluyen la directiva para que se cargue el módulo en cuestión mientras que los archivos .conf incluyen directivas de configuración que únicamente se aplicarán si se carga un módulo determinado. Esto se consigue con la directiva *IfModule* que funciona como un if de programación.

Realmente lo que encontramos en el directorio son enlaces simbólicos a los archivos en */etc/apache2/mods-available*

mod_status y mod_info

Estos dos módulos nos permiten obtener información muy útil sobre nuestro servidor.

mod_status está activado por defecto en nuestra distribución de Apache, pero se activa y desactiva como cualquier otro módulo. Podemos ver y modificar su configuración en */etc/apache2/mods-available/*

```
gedit status.conf
```

Vamos a aprovechar para ver otra directiva de configuración. Si nos fijamos ahora solo nos permite consultar el estado desde nuestra máquina.

```
Require local
```

Podemos añadir una línea debajo con la dirección IP de otra máquina para que nos dé acceso desde ella.

Se consulta dicha información en la dirección o nombre de nuestro servidor con: http://192.168.56.101/server-status si nuestra dirección fuera esa.

mod_info no viene activado por defecto

```
sudo a2enmod info
```

podemos consultar o modificar su configuración

```
gedit /etc/apache2/mods-enabled/info.conf
```

Esta información se puede consultar en http://192.168.56.101/server-info modificando la dirección IP por la tuya o el nombre de tu servidor.

Recuerda que para poder usar los cambios en la configuración hay que reiniciar el servidor.

Consulta los dos archivos de configuración asociados al módulo status. Estudia para qué se utilizan las directivas que se ejecutan en caso de que se cargue el módulo y coméntalo en clase.

Habilita ambos módulos y permite que se consulten desde la máquina anfitrión.

Prueba el módulo mod_status. Comprueba que funciona y activa la opción que muestra el estado extendido.

Activa el módulo mod_info y comprueba que funciona. Añade a la información estándar información sobre el módulo mod_status.

Prueba a consultar ambas páginas de información y echa un vistazo a qué información se puede obtener.

Directorios personales de usuarios

En determinadas circunstancias (por ejemplo una institución educativa como un instituto o una universidad) es útil que cada usuario tenga un directorio en el que pueda crear su propio conjunto de páginas web. Si el número de usuarios es grande, esto puede cargar innecesariamente de trabajo al administrador del servidor. Apache ofrece una alternativa automatizada para esta situación mediante el módulo *mod_userdir*.

Si activamos el módulo (por defecto no viene activo) cada usuario tendrá un espacio al que se accederá mediante la URL http://sitioejemplo.com/~nombreusuario donde nombreusuario será el nombre que tiene en el servidor Linux.

En la distribución que usamos ya existe en *mods-available/* una configuración para este módulo así que en la explicación de la siguiente directiva no es necesario escribirlo.

```
<IfModule mod_userdir.c>
    UserDir public_html
    UserDir disabled root

    <Directory /home/*/public_html>
        AllowOverride FileInfo AuthConfig Limit Indexes
        Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
        <Limit GET POST OPTIONS>
            Require all granted
        </Limit>
        <LimitExcept GET POST OPTIONS>
            Require all denied
        </LimitExcept>
    </Directory>
</IfModule>
```

UserDir

Indica si se debe permitir que cada usuario de nuestro sistema tenga su propia carpeta personal en el servidor web y establece cuál será la ruta desde el servidor para acceder a dicha carpeta. Esto tiene sentido por ejemplo si se monta un servidor Apache en un instituto y cada profesor tiene su propio usuario. Para que cada uno tenga su propia web personal se puede activar esta opción.

```
UserDir public_html
```

Es la opción más frecuente ya que crearía en nuestro servidor rutas para cada usuario. Si por ejemplo hay un usuario Sergio y otro Maria tendríamos las rutas:

```
www.servidordeprueba.es/~Sergio
www.servidordeprueba.es/~Maria
```

Y cada usuario tendría en su carpeta personal un subdirectorio public_html para publicar lo que quisiera. Como se puede ver el argumento detrás de *UserDir* indica cómo se llamará ese subdirectorio.

```
UserDir disable
```

Es la opción por defecto y hace que los usuarios no tengan su propio espacio.

En caso de activar los directorios de usuarios se recomienda deshabilitar el de *root*

```
Userdir disabled root
```

Cada usuario puede crear ahora en su carpeta personal un directorio *public_html* donde ubicar sus páginas. Aquí se explica en más detalle cómo se utiliza la carpeta personal.

No se activa esta opción para el usuario *root*.

> Este tipo de configuración se ha utilizado mucho por ejemplo en universidades donde cada profesor tenía una página web disponible solo con tener un usuario. Todavía no puedes probar la configuración de esta directiva en tu servidor, pero sí leer la documentación de la directiva ¿Qué usuarios tienen ahora carpeta personal? La ruta por defecto a la carpeta personal no es muy fácil de escribir con un teclado en muchos países, ¿cómo la cambiarías? Teniendo en cuenta un servidor remoto, ¿qué inconveniente ves a esta forma de funcionar? Si tuvieras que plantear alguna funcionalidad similar en la actualidad, ¿cómo lo harías?

En la documentación del módulo puedes observar que el directorio en el que cada usuario podrá crear sus páginas es altamente configurable.

> Activa el módulo mod_userdir y crea una carpeta para el usuario con el que instalaste Linux. Dentro de la carpeta crea un archivo prueba.html y configura el módulo para que al acceder a la carpeta personal del usuario desde un navegador web se muestre dicho archivo por defecto.
>
> Por último desactiva el módulo.

El módulo MIME

Los tipos MIME sirven para identificar tipos de archivos. Este módulo se usa para ello y algunas otras cosas. Existe una configuración ya hecha en *mime.conf*.

TypesConfig

Indica dónde estará el archive que describe los tipos MIME (mime.types) o su equivalente si lo hemos cambiado. No hay muchas razones para ello, por lo que deberíamos dejar el valor por defecto.

```
TypesConfig conf/mime.types
```

DefaultType

Establece el tipo MIME para todos aquellos archivos a los que no se les pueda asignar uno mediante su extensión, etc. Si tenemos un servidor donde la mayoría del contenido son páginas HTML, XML, etc es buena idea usar el valor por defecto, pero si la mayoría del contenido son archivos binarios (fotografías, programas, etc) sería conveniente establecer *application/octet-stream*.

```
DefaultType text/plain
```

Se puede especificar más:

```
DefaultType text/html
```

Investiga en Internet qué son los tipos MIME y cuáles hay. ¿Qué organismo se encarga de gestionarlos? ¿De qué otras cosas se encarga?

AddEncoding

Especifica un tipo particular de codificación para determinadas extensiones de archivos. También se puede usar *AddEncoding* para indicar a los navegadores que descompriman ciertos archivos mientras los descargan:

```
AddEncoding x-compress Z
AddEncoding x-gzip gz
```

AddLanguage

Sirve para asociar extensiones a idiomas determinados para el contenido. Esta directriz es útil para la negociación de contenidos entre el servidor y el navegador web del cliente ya que Apache puede devolver contenidos en diferentes idiomas dependiendo de la configuración del idioma del navegador Web. Es útil sobre todo en los casos en los que tenemos el sitio escrito en varios idiomas ya que permitirá que Apache sirva la adecuada en cada caso en función de la configuración del navegador del cliente. Esto se realiza de forma transparente para el usuario final.

Los códigos de idiomas se definen en la especificación ISO 639. Más concretamente en la ISO 639-1 los códigos de dos letras y en la ISO 639-2 los de tres letras que incluyen más idiomas.

```
# Danés (da) - Holandés (nl) - Inglés (en) - Estonio (et)
# Francés (fr) - Alemán (de) – Griego Moderno (el)
# Italiano (it) - Noruego (no) - Koreano (kr)
# Portugués (pt) - Luxemburgués(ltz)
```

```
    # Español (es) - Sueco (sv) - Checo(cz)
    # Polaco (pl) - Portugués de Brasil (pt-br) - Japonés (ja)
    # Ruso (ru) - Croata (hr)
    #
    AddLanguage es .es
    AddLanguage da .dk
    AddLanguage nl .nl
    AddLanguage en .en
    AddLanguage et .et
    AddLanguage fr .fr
    AddLanguage de .de
    AddLanguage el .el
    AddLanguage it .it
    AddLanguage ja .ja
    AddLanguage pl .po
    AddLanguage kr .kr
    AddLanguage pt .pt
    AddLanguage no .no
    AddLanguage pt-br .pt-br
    AddLanguage ltz .ltz
    AddLanguage sv .se
    AddLanguage cz .cz
    AddLanguage ru .ru
    AddLanguage tw .tw
    AddLanguage hr .hr
```

Ten en cuenta que el indicador del idioma no tiene por qué ser idéntico al sufijo como se ve en varios de los ejemplos.

AddCharset

Es igual que *AddLanguage* pero para añadir nuevos juegos de caracteres.

```
    AddCharset ISO-8859-1 .iso8859-1 .latin1
    AddCharset ISO-8859-2 .iso8859-2 .latin2 .cen
```

Hosts virtuales

Apache puede servir varios sitios web desde un único servidor web. El cliente nunca diferenciará si son sitios en servidores diferentes o en la misma máquina. Apache es un servidor muy potente para la utilización de esta opción.

Si por ejemplo somos los gestores de dos dominios DNS (www.laempresa.es y www.mipagina.es) podemos alojar ambos sitios en el mismo servidor Apache. Uno de estos dominios se considerará el sitio principal y todos los demás serán los hosts virtuales.

Ventajas:

1. Aprovechar el hardware existente.
2. Aprovechar las direcciones IP públicas disponibles.

Una gran ventaja en el uso de hosts virtuales en Apache es que **permite heredar** la configuración del sitio principal por lo que no habrá que reconfigurar todas las directivas, solo las que cambien.

Al igual que pasaba con los módulos, existen dos directorios para contener los sitios virtuales, uno para los disponibles y otro para los activos: */etc/apache2/sites-available* y */etc/apache2/sites-enabled* respectivamente. El segundo contiene enlaces simbólicos a los sitios del primero que estén activos.

Esta documentación se basa ya en la versión 2.4 de apache, pero como ayuda a los que hayan usado la versión 2.2 o se encuentren sitios web configurados con dicha versión, dejo aquí unas indicaciones de los cambios a realizar. Realmente ninguno es directamente de Virtual Hosts, sino que son directivas generales que suelen usarse en este ámbito.

Sitio por defecto

La configuración del servidor virtual por defecto se puede consultar

```
gedit /etc/apache2/sites-available/000-default.conf
```

lo que nos mostrará

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
```

```
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf
    </VirtualHost>
```

Podemos comprobar que en el directorio de sitios activos se encuentra un archivo *000-default* que es el enlace simbólico al sitio por defecto.

> Estudia y comenta en clase las directivas incluidas en la configuración del sitio virtual por defecto.

En el archivo podemos observar que el directorio raíz es */var/www/html* y la configuración que tiene el directorio. Ten en cuenta que es necesario reiniciar Apache para que los cambios en la configuración surtan efecto.

Nota: hasta la versión 2.2 el directorio por defecto era */var/www* y dicha ruta sigue siendo muy habitual.

Sin embargo en el archivo de configuración principal de Apache, apache2.conf, nos encontramos:

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

Un directorio hereda las directivas del superior si no se sobrescriben. Por ello, todos los directorios que creemos dentro de */var/www/* tendrán la misma configuración que el raíz excepto en aquellas directivas que especifiquemos en la configuración propia del directorio.

Vamos a modificar la configuración del sitio por defecto. Crea un archivo *DAW.html* en el directorio raíz con tu nombre y tus datos. Ahora queremos que el índice o página principal del directorio sea la que acabas de crear. ¿Qué directiva debes añadir? ¿Es posible establecer más de una página principal? En caso afirmativo, ¿cuál se cargará si accedemos al sitio sin especificar página? ¿Qué sucede si no hay ninguna de las páginas establecidas como página principal? Busca una directiva que deshabilite el listado de archivos incluso si no encuentra ninguna de las páginas por defecto.

Crea un nuevo directorio dentro del directorio raíz. El nombre del directorio será tu apellido. Crea varios archivos html pero no establezcas ninguno como el principal. Accede al directorio desde un navegador. Ahora modifica la configuración del directorio para que liste el contenido. ¿Qué sucede? Establece uno de los archivos como el principal para el directorio y vuelve a probar.

Recuerda que después de cada cambio en la configuración es necesario reiniciar Apache.

Como ya debes haber visto en la práctica, para deshabilitar el listado de los contenidos de un servidor, host virtual o directorio se usa

```
Options –Indexes
```

Si quieres activarlo para un servidor, directorio o host virtual se utiliza

```
Options Indexes FollowSymLinks
```

La opción de seguir los enlaces no es obligatoria pero se suele añadir.

Ten en cuenta que hay que mirar que la configuración no esté sobrescrita en la configuración del host virtual por defecto o de algún directorio en particular. Lo que debe hacerse en estos casos es una comprobación de la directiva de lo particular a lo general: directorios, host virtual y servidor.

Modificando los mensajes de error

La directiva *ErrorDocument* sirve para establecer mensajes de error personalizados para diferentes situaciones. Los códigos de error que pueden utilizarse son los definidos por la organización w3 para el protocolo http en este caso la versión 1.1 y se pueden consultar aquí. En esta otra página se explican de una manera más comprensible. Ten en cuenta que el código es solo el último número de cada apartado y tiene tres cifras. Esta directiva se usará tantas veces como se crea necesario. Se establece de manera independiente **para cada sitio virtual** y se suele escribir debajo de *CustomLog*.

En el apartado anterior hemos visto que se usa una variable de entorno llamada *APACHE_LOG_DIR* para determinar el directorio donde se ubicarán los archivos de log. En esta instalación es */var/log/apache2/* pero si no podemos buscar los archivos en la estructura de directorios o consultar el valor de la variable de entorno.

Establece acciones personalizadas para el error 404. Primero redirige las consultas a páginas no encontradas a la página principal del sitio. Luego prueba a redirigir estos errores a la página principal del instituto. En la tercera prueba debes mostrar un mensaje directamente que diga “página no encontrada, consulte a Nombre Apellido” con tu nombre y apellido. Por último crea un archivo html que explique el error y haz que se muestre si se da el caso.

Consulta la lista de errores definidos y apunta los que creas más útiles. Coméntalos en clase.

Alias a otros directorios

Por defecto la estructura de un sitio web será la que demos a los directorios desde el punto de montaje del directorio raíz del sitio, sin embargo es posible incluir otros directorios del árbol de nuestro servidor y hacer que parezcan parte del sitio como cualquier otra carpeta que esté contenida físicamente dentro. Es algo similar a tener un enlace simbólico a archivos o carpetas en otro directorio.

En el archivo de configuración del sitio principal visto antes ya existe un alias a un directorio que se encuentra fuera de la carpeta definida para el sitio.

Para realizar esta operación se utiliza la directiva *Alias* y justo debajo (no es obligatorio pero sí una buena práctica) se configura con la directiva *Directory* como cualquier otro directorio.

En la guía rápida de directivas, se pueden aplicar a directorios todas las que llevan “d” en la tercera columna.

> Crea un directorio /var/extras y coloca dentro una página HTML que te permita diferenciarla de las demás. Configura el directorio para que aparezca como /extras en nuestro sitio web y aplica las directivas de directorio que consideres importantes.

Redirecciones

Algunas veces es útil poder redirigir las llamadas a una dirección web para que se procesen en otro punto. Por ejemplo si hemos cambiado la estructura de nuestro sitio y hemos reubicado la página de contacto podemos querer que los usuarios que accedan mediante la antigua dirección sean redirigidos automáticamente a la nueva. También puede ser útil si hemos dividido nuestro sitio para facilitar la gestión o el mantenimiento.

Para ello se usa la directiva *Redirect* que nos permite asociar una dirección absoluta o relativa a otra.

> Crea una redirección en el sitio principal del servidor para que en caso de que alguien quiera acceder a /aficiones se redirija a una página que visites muy a menudo.
>
> Crea ahora una redirección que en el caso de alguien quiera acceder a /ext se le muestre el contenido de /extras que creaste en el punto anterior.

Creación de hosts virtuales

Existen tres formas diferentes de crear hosts virtuales en Apache:

1. Basados en **nombres**: Este tipo es la opción más común. Se configura todo para que múltiples dominios DNS apunten a una única máquina con Apache. Requiere configuración del servidor DNS para que funcione. Este método hace muy fácil migrar nuestro servidor a otra dirección IP. Es el **único método que vamos a estudiar**.
2. Basados en **IP**: En este método se necesita configurar las direcciones IP de cada sitio en Apache. El servidor físico tendrá varias direcciones IP, una para cada sitio.
3. Basados en **puertos**. Cada sitio se atenderá en la misma IP o nombre pero en distintos puertos. Es una extensión de cualquiera de las alternativas anteriores.
4. Varios **servidores principales**: En esta opción se mantienen varias configuraciones principales en el servidor. Solo se recomienda su uso si es necesario tener un archivo de configuración diferente para cada sitio. Apenas se usa y es la opción menos recomendada para configurar nuestro servidor.

El uso de estas alternativas no es excluyente. Pueden combinarse varias o todas en el mismo servidor Apache.

En esta dirección está la documentación sobre *hosts* virtuales de apache. En esta otra se pueden ver ejemplos de configuración.

> Estudia y comenta en clase los primeros ejemplos del enlace anterior. Céntrate solo en los primeros, que tratan de los tipos listados anteriormente.

Las directivas necesarias para crear *hosts* virtuales se encuentran en el módulo *core* de Apache por lo que no es necesario activar ningún módulo.

Para usar estos métodos es necesario poder hacer que todos los nombres de dominio asociados a los *hosts* virtuales que vamos a crear apunten a la dirección IP o las direcciones IP de nuestro servidor web. Si hemos contratado un nombre de dominio con un proveedor éste nos proporcionará la manera de hacerlo.

En caso de que tengamos o queramos publicar nuestros propios sitios debemos configurar un servidor DNS. En los apéndices encontrarás una manera muy simple de configurar DNS en nuestro servidor. Otra opción para probarlo es editando el archivo *hosts*

```
sudo gedit /etc/hosts
```

y en él añadir los alias necesarios.

Los sitios virtuales utilizan una estructura de directorios similar a la que hemos visto para los módulos. En

```
/etc/apache2/sites-available/
```

Encontramos los sitios disponibles, que al ser activados crean un alias en

```
/etc/apache2/sites-enabled/
```

Hosts virtuales basados en nombres

Este método es el más recomendado ya que requiere una única dirección IP para poder alojar múltiples sitios. Podemos ver la documentación específica de Apache.

NOTA: En Apache, hasta la versión 2.2 era necesario añadir lo siguiente.

Podemos comprobar que en el archivo */etc/apache2/ports.conf* se encuentra la directiva

```
NameVirtualHost *:80
```

Esta directiva es la necesaria para poder usar los hosts virtuales basados en nombres. Al activarse deshabilita el servidor principal (el que habíamos usado en la instalación manual) y por ello en este tipo de instalación el servidor virtual es realmente un sitio virtual por defecto.

En caso de recibir un **error "*Apache2 NameVirtualHost *:80 has no VirtualHosts*"** suele ser porque has puesto la directiva NameVirtualHost en ambos archivos de configuración de los sitios. Al poder ser una directiva de servidor, con ponerla en el *httpd.conf* vale.

FIN NOTA.

El primer paso es crear los registros DNS para que el nombre de dominio apunte a nuestra máquina o añadirlos en el fichero *hosts*.

Debemos crear un directorio para cada sitio y así tener los documentos separados. Además habría que crear un archivo *index.html* en el directorio para que lo cargue al consultar el sitio.

```
mkdir /var/www/ejemplo2.es
```

En este punto podemos acceder a los datos de ejemplo2 mediante la dirección http://localhost/ejemplo2.es/ pero nuestro objetivo es poder hacerlo escribiendo http://ejemplo2.es/

Lo tercero que deberíamos hacer es escribir los datos del nuevo host virtual en la configuración de Apache. Recuerdo que hemos visto dos casos diferentes de dónde se encontraba la configuración de Apache, pero con la opción que estamos usando actualmente la configuración principal estaba en el archivo */etc/apache2/apache2.conf* y desde él se importaban otros muchos. Podemos ver que se importa una carpeta completa de la que coge la configuración de los hosts virtuales.

```
…
# Include the virtual host configurations:
Include sites-enabled/
…
```

Sin embargo, antes de hacer esto debemos crear el host virtual en la carpeta de hosts disponibles y lo activaremos cuando esté terminado. La carpeta en la que podemos añadir los datos de nuestros hosts virtuales es */etc/apache2/sites-available*

Si vamos a la carpeta que nos indican podemos consultar el archivo de la configuración de un host virtual, el sitio por defecto. Esto puede servirnos ejemplo y guía para crear otros.

Creamos el archivo de configuración

```
sudo gedit /etc/apache2/sites-available/ejemplo2.es
```

y escribimos lo siguiente

```
<VirtualHost 10.0.2.15:80>
     ServerName ejemplo2.es
     ServerAlias www.ejemplo2.es
     ServerAdmin alguien@ejemplo2.es
     DocumentRoot /var/www/ejemplo2.es
     #
     # Aqui pueden ir otras directivas.
     # Por defecto hereda las del archivo principal.
     #
</VirtualHost>
```

Entre todas las directivas que se pueden incluir son especialmente útiles las referentes a **directorios** ya que si el sitio consta de varios (lo más habitual) puede interesarnos tener configuraciones diferentes para cada uno.

También es habitual configurar registros de error independientes para cada sitio mediante la directiva adecuada. Esto se hace dentro de la configuración de cada sitio virtual dejando el principal para el servidor en si. Ya vimos que en el sitio por defecto se incluían.

```
     ErrorLog ${APACHE_LOG_DIR}/error.log
     CustomLog ${APACHE_LOG_DIR}/access.log combined
```

Pero por supuesto debemos **dar otros nombres a nuestros registros** para que no se usen los mismos.

Es buena idea **revisar la configuración del sitio principal** para hacernos una idea de qué configurar ya que en el ejemplo anterior he reducido las directivas al mínimo.

Si en lugar de la dirección IP escribimos * se realizará para todas las IP que tenga la máquina.

En los dos casos en los que hemos escrito la dirección IP y el puerto podríamos haber puesto un asterisco (*) en lugar de la dirección pero se estaríamos indicando a Apache que escuche todas las peticiones lo que podría generar conflictos. Siempre es preferible concretar los datos.

Ahora ya hemos configurado el nuevo host así que podemos activarlo.

```
sudo a2ensite ejemplo2.es
```

y recargar los sitios de Apache

```
service apache2 reload
```

si al hacerlo nos da el error

```
* Reloading web server config apache2                                    apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName
```

Debemos abrir el archivo *httpd.conf* y añadir

```
ServerName LocalHost
```

O el nombre completo de nuestro servidor.

Ya podemos acceder a nuestro documento mediante www.ejemplo2.es

Para desactivar un sitio virtual se hace con (por ejemplo para desactivar el sitio por defecto).

```
sudo a2dissite default
```

Crea un nuevo host virtual basado en nombre y prueba que todo sea correcto. Crea otro y verás que ahora tienes la posibilidad de acceder a tres sitios en la misma máquina. Haz la asociación mediante el archivo *hosts* de Linux. Posteriormente eliminaremos la entrada del archivo hosts y realiza la asociación mediante un servidor DNS. Por último desactiva el sitio virtual.

Para esta práctica vas a necesitar configurar tu servidor DNS. En los apéndices explico cómo.

Control de acceso

En este punto comenzamos con varios aspectos relacionados con la seguridad de nuestro servidor web.

Con este primer apartado vamos a considerar los aspectos que nos van a permitir filtrar el acceso a determinados recursos.

El control de acceso se refiere a cualquier método que nos permita filtrar el acceso a algún recurso determinado. Hay principalmente dos módulos implicados en el control de acceso en Apache: mod_auth_core y mod_authz_host aunque para cubrir todos los aspectos se usan también mod_setenvif y mod_rewrite.

También existen tres métodos para gestionar el control de acceso relacionados con estos módulos. El control de acceso está íntimamente relacionado con la autorización y autenticación que veremos en el siguiente punto.

Control de acceso basado en la dirección

Este tipo de control se basa en el uso del módulo mod_authz_host y en las direcciones IP de las máquinas que quieran acceder a nuestro servidor. Podemos comprobar que esté módulo ya está activo en nuestra instalación y de hecho ya hemos usado anteriormente este tipo de control de acceso para permitir acceder a partes de nuestro servidor desde la máquina anfitrión en las prácticas.

Ya comentamos al principio del tema que este para este caso las directivas que se utilizaban **hasta la versión 2.2** eran *Allow* y *Deny*. Generalmente van asociadas con otra, *Order*.

Sin embargo actualmente todo se lleva a cabo con la directiva *Require*. En la propia documentación podemos encontrar muchos ejemplos y diferentes usos pero este cambio ha simplificado enormemente la tarea. Sin embargo debemos centrarnos primero en comprender cómo funciona la directiva de forma más genérica estudiando la documentación del módulo moth_auth_core.

Ten en cuenta que la misma directiva nos sirve para permitir o denegar acceso mediante el uso de *granted* o *denied*.

> Investiga cómo se puede filtrar un dominio entero y un rango de direcciones IP en lugar de máquinas sueltas.
>
> ¿es posible filtrar direcciones con una máscara que no sea múltiplo de 8? ¿cómo?
>
> ¿Para qué sirven los contenedores? Estudia RequireAll, RequireAny y RequireNone.
>
> Crea un directorio con tu nombre en el sitio por defecto. Permite el acceso desde la máquina anfitrión pero deniégalo desde la máquina en la que se encuentra el servidor.

Ten en cuenta que estas directivas no son exclusivas de los directorios. Pueden aplicarse al sitio web completo por ejemplo. Para ello irán el la configuración general del sitio y no dentro de unas etiquetas *Directory*.

Control de acceso por variable de entorno

Este tipo de control de acceso se realiza mediante el uso de los módulos mod_authz_host y mod_setenvif. Se basa en permitir el acceso según la configuración de alguna variable de entorno de la máquina del usuario y por ello no es muy recomendable.

Control de acceso con el módulo rewrite

Mediante el uso del módulo mod_rewrite podemos controlar el acceso según criterios arbitrarios. Por ejemplo si queremos denegar el acceso durante el periodo de las ocho de la tarde a las 6 de la mañana, escribiremos

```
RewriteEngine On
RewriteCond %{TIME_HOUR} >20 [OR]
RewriteCond %{TIME_HOUR} <07
RewriteRule ^/fridge - [F]
```

El uso de este método se basa en las directivas RewriteCond y RewriteRule pero queda fuera del alcance de este curso y se indica solo como introducción a las posibilidades que permite.

Autenticación y autorización

Estos dos términos van ligados pero no son lo mismo a pesar de que mucha gente los confunde. La autenticación consiste en comprobar que alguien es quien dice ser mientras que la autorización es comprobar que alguien tiene permiso para acceder a un lugar o recurso determinado.

Por ejemplo, si quieres viajar al extranjero (fuera de los países que ha firmado el Acuerdo de Schengen) necesitas un pasaporte y un visado. El pasaporte es un documento general que sirve para demostrar que eres quien dices ser mientras que el visado te autoriza a visitar un país determinado.

En informática, la autenticación puede darnos acceso a diferentes recursos para los que estemos autorizados, e incluso estas autorizaciones pueden variar dependiendo de diferentes circunstancias. Por ejemplo, podemos tener permiso para acceder a determinado recurso en una franja horaria determinada o desde la oficina de trabajo pero no desde casa.

El proceso de autorización suele implicar la autenticación. Por ejemplo si un cliente quiere acceder a determinado recurso el servidor le pide que se autentique (mediante un mensaje de estado 401: Authoritation Required) por ejemplo con una solicitud de usuario y contraseña. Si esta autenticación es positiva se permitirá el acceso y si no se responderá con otro mensaje 401.

Un problema de esta forma de autenticación es que la contraseña ni se encripta ni se oculta, por lo que más adelante hablaremos del protocolo *https*. Cualquiera con un *sniffer* podría interceptar los nombres de usuario y contraseñas.

En Apache podemos centrarnos en la documentación sobre estos conceptos. Vamos a **ir viendo todo el enlace paso a paso** y entendiendo toda la información que nos proporcionan. Es especialmente interesante el apartado en el que se profundiza en *Require*.

El comando *htpasswd* es de los primeros que aparecen. Investiga cómo se usa y qué hace. Desde esta versión de Apache es necesario instalarlo aparte ya que no está incluido en la instalación general.

sudo apt-get install apache2-utils

El comando puede encriptar las contraseñas de maneras diferentes.

En Apache desde la versión 2.3 la base de autenticación se ha centralizado en mod_authn_core y la de autorización en mod_authz_core. El número de módulos implicados realmente es mucho mayor, pero se agrupan en tres grandes conjuntos: Uno para la autenticación, otro para la autorización y otro para determinar contra qué sistema se comprobarán las credenciales en la autenticación (por ejemplo un archivo, una base de datos, etc).

Los módulos mod_auth_core y mod_auth_basic

Estos módulos nos permite realizar autorización de una manera bastante básica. Nos van a permitir establecer acceso mediante usuario y contraseña a secciones de nuestro sitio. Básicamente es lo que hemos comentado en el enlace anterior.

Por ejemplo podemos establecer un directorio al que haya que acceder con contraseña. Podemos elegir cualquier directorio siempre que el **usuario** que lanza apache (directiva *User*) tenga acceso a él. No es necesario que el directorio esté en la estructura básica de nuestro sitio, pero si no está tendremos que establecer un *Alias*.

```
mkdir /var/www/privado
mkdir /var/secreto
```

Ahora añadimos al sitio en el que queramos gestionar la autenticación

```
gedit /etc/apache2/sites-available/000-default
```

la configuración para el directorio

```
<Directory "/var/www/privado">
        AuthName “Acceso privado: Introduzca su usuario y contraseña”
        AuthType  Basic
        AuthUserFile /var/secreto/.miembros
        Require valid-user
</Directory>
```

- *AuthName* le indica al usuario qué hacer. Es un mensaje para el usuario.
- *AuthType* es el tipo de autenticación que usaremos; http solo admite ***Basic***. La otra opción que existe es ***Digest*** que a diferencia de la opción *Basic* no transmite los nombres de usuario y contraseña como texto plano (y por lo tanto es una opción de seguridad mejor) pero que no está disponible para todos los navegadores Web como opción “out-of-the-box”. El cifrado que usa la opción *Digest* es bastante débil por y aunque su uso no es idéntico al de la opción *Basic* es bastante similar, por lo que no lo veremos.
- *AuthUserFile* es el archivo que se utilizará para guardar las contraseñas. Indica la ruta y se llamará .miembros.
- *Require* especifica que será necesario acceder con un usuario válido.

Ahora necesitamos crear el archivo de contraseñas.

```
htpasswd -c /var/secreto/.miembros srsergio
```

Donde deberás especificar la ruta a los ejecutables de apache si no es la misma, -c es la opción para crear el archivo así que si vas a añadir otro miembro debes quitarla. La ruta al archivo de contraseñas debe ser la misma especificada para la configuración del sitio y *srsergio* será el nombre de usuario que queremos crear. Nos pedirá que creemos una contraseña.

Podemos asegurarnos de que se ha creado el usuario abriendo el archivo. Date cuenta de que la contraseña está codificada.

```
gedit /var/secreto/.miembros
```

Reinicia apache

```
apachectl restart
```

Ahora ya tienes una sección para acceso solo con usuario y contraseña.

Ten en cuenta que la debilidad de **http** en la transmisión de la información hace que este método **no sea seguro** pero se puede solucionar usando **https** como veremos más adelante.

Crea una sección privada en tu sitio e introduce algunos archivos html. Crea al menos dos usuarios que tengan acceso y prueba a acceder tanto con ellos como con datos no válidos.

Siempre se habla de que la información que se transmite por Internet como texto plano es muy fácil de interceptar. ¿Cómo se haría esto es nuestra red interna? ¿Qué programas necesitas? ¿Serías capaz de interceptar un usuario y contraseña enviados por un usuario para acceder a la sección privada creada en el punto anterior?

Ampliación: Investiga cómo se haría lo mismo con la opción *Digest*. Parece ser que Apache trae Digest desactivado por defecto y lo único que hace es codificar pero no encriptar los datos por lo que no merece la pena ni activarlo.

Los ficheros .htaccess

Aunque los hemos usado de manera intuitiva, es buen momento para profundizar en los archivos de configuración de Apache en los que podemos usar diferentes secciones.

Las soluciones vista hasta ahora no son muy adecuadas si queremos poder delegar la creación y control de zonas privadas para miembros determinados. Esto puede ser muy útil si por ejemplo hemos montado una web para una empresa que tiene su propio administrador de sistemas ya que evitará que tengamos que gestionar todo nosotros. También nos facilitará el trabajo si hay muchos cambios en las zonas privadas o miembros que se conecten a ellas.

Podemos conocer más sobre *htaccess* en este artículo. Otra web nos introduce a algunos de los usos más comunes de estos archivos con una segunda parte.

Para permitir el uso de ficheros *.htaccess* en nuestro servidor o sitio virtual (la directiva se puede usar en ambos entornos) lo primero que debemos hacer es modificar el archivo de configuración. En mi caso voy a hacerlo en el sitio virtual por defecto.

Debemos modificar la directiva

```
    AllowOverride None
```

Y cambiarla a

```
    AllowOverride AuthConfig
```

El lugar en el que modificar la directiva depende de lo que necesitemos. Ten en cuenta que las directivas se heredan si no se encuentra otra más específica, por ello, en nuestro archivo de configuración del sitio virtual por defecto debería ir en el directorio /var/www por lo menos ya que si lo ponemos en el raíz pero no en el primero, se mantendría la configuración anterior.

```
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride AuthConfig
        Require all granted
    </Directory>
```

Lo que permite que modifiquemos las directivas de autorización mediante un fichero *.htaccess*. En muchos sitios indican que hay que permitir la sobre escritura de todas las directivas mediante *AllowOverride All* pero es evidente que es peor opción.

Luego reiniciamos el servidor Apache

```
    apachectl restart
```

Ahora podríamos crear ficheros directorios y configurar su control de acceso en cada uno de ellos.

```
    mkdir /var/www/ficheros/
    cd /var/www/ficheros/
```

En cada directorio que queremos gestionar así debemos crear un fichero *.htaccess* y darle un contenido similar al siguiente.

```
AuthName “Sección Privada: Prueba de .htaccess”
AuthType  Basic
AuthUserFile /var/secreto/.miembros
Require valid-user
```

Ten en cuenta que:

- Lo que hemos hecho ha sido sacar la configuración del control de acceso del archivo de configuración del servidor/sitio virtual a un archivo independiente.
- He usado el mismo archivo de usuarios y contraseñas que en el punto anterior para darle coherencia a los ejemplos, pero esto no es necesario.
- Los usuarios y sus contraseñas se crearían igual que en el apartado anterior.
- La creación o modificación de un fichero de este tipo no implica reiniciar el servidor.
- Se debe mejorar la seguridad cambiando los permisos de acceso al fichero *.htaccess*. Por lo menos el usuario de Apache debe tener acceso.

Crea dos directorios en tu sitio por defecto y configúralos para que en el primero puedan acceder dos usuarios y en el segundo solo uno de ellos. Para ello en *Require* debe ir su nombre de usuario.

Agrupando usuarios para el control de acceso

Si queremos refinar más el control de acceso tenemos varias alternativas. Por ejemplo podemos permitir el acceso a solo algunos usuarios especificando sus nombres en la cláusula *Require* en cualquier sitio que la usemos.

```
require user sergio maria
```

Otra alternativa es crear archivos de usuarios diferentes para distintos directorios de nuestro sitio.

```
<Directory "/var/www/ventas">
        AuthName “Acceso privado: Introduzca su usuario y contraseña”
        AuthType  Basic
        AuthUserFile /var/secreto/.miembros-ventas
        Require valid-user
</Directory>

<Directory "/var/www/finanzas">
        AuthName “Acceso privado: Introduzca su usuario y contraseña”
        AuthType  Basic
        AuthUserFile /var/secreto/.miembros-finanzas
        Require valid-user
</Directory>
```

La última opción pasará por usar la directiva *AuthGroupFile*. Así podremos tener todos los usuarios en el mismo fichero y luego asignarles grupos en otro archivo.

```
<Directory "/var/www/ventas">
        AuthName “Acceso privado: Introduzca su usuario y contraseña”
        AuthType  Basic
        AuthUserFile /var/secreto/.miembros
        AuthGroupFile     /var/secreto/.grupos
        Require group ventas
</Directory>
```

```
<Directory "/var/www/finanzas">
        AuthName “Acceso privado: Introduzca su usuario y contraseña”
        AuthType  Basic
        AuthUserFile /var/secreto/.miembros
        AuthGroupFile     /var/secreto/.grupos
        Require group finanzas
</Directory>
```

Ten en cuenta que:

- Se usa el mismo archivo de miembros para todos los directorios.
- Hay que añadir la directiva y el fichero de grupos.
- En la cláusula Requiere se indica el grupo que puede acceder.

El archivo .grupos será un fichero de texto con algo como

```
ventas: sergio maria
finanzas: carlos roberto
```

Modifica la configuración del ejercicio anterior para que puedan acceder dos grupos de usuarios diferentes a cada directorio con el último método. ¿Puede un usuario estar en más de un grupo?

Añade un tercer directorio y otro grupo. Da acceso al tercer directorio a dos de los tres grupos. ¿Cómo se hace esto?

El protocolo HTTPS

Ya hemos visto que el protocolo HTTP es muy inseguro para transmitir información sensible. Es muy fácil capturar esta información y leer datos como nombres de usuarios o claves. Cualquier información que se envía va sin cifrar por lo que se transmite como un texto que cualquiera que capture nuestra comunicación puede leer.

El protocolo HTTPS se utiliza para evitar este problema. Sus siglas se corresponden con *Hypertext Transfer Protocol* ***Secure*** lo que ya indica que añade seguridad a HTTP. Realmente no es un protocolo sino que se vasa en introducir una capa con el protocolo SSL (o derivados, más adelante se habla de todo esto) entre la capa de transporte y la capa de aplicación en el conjunto de protocolos TCP/IP.

El protocolo HTTPS proporciona **autenticación** de los usuarios con el servidor con el que se conecta. Por ejemplo cuando nos conectamos a nuestro correo web (por ejemplo Gmail) suele hacerse mediante una conexión HTTPS.  Con el uso de este protocolo también se consigue **encriptación** de la información que enviamos y recibimos. Por ello cuando nos conectamos a un servidor que solo utiliza HTTPS para la autenticación el navegador nos avisa (una vez nos hemos conectado) de que el resto de la comunicación no usa HTTPS y por lo tanto la trasferencia de información puede ser insegura. Esto nos protege contra los ataques de tipo *Man-In-The-Middle*. Todo lo que se envían en un mensaje HTTPS está encriptado, incluyendo las cabeceras. Por supuesto la encriptación también está sujeta a ataques que intenten descifrar la clave y el algoritmo usados para cifrar el contenido.

> Investiga qué algoritmos y qué tipo de claves se utilizan en Internet actualmente.
>
> ¿Qué son los algoritmos de clave simétrica y los algoritmos de clave asimétrica? ¿Qué con las claves privadas y claves públicas? ¿Cómo se utilizan?

> Esta página de la Wikipedia te puede ayudar a comprender estos conceptos y los que veremos a continuación. Necesitarás tener una idea de los puntos anteriores de este ejercicio para entenderla.

Un sitio que use HTTPS debería tener todos sus contenidos protegidos por este protocolo para evitar posibles ataques o robos de información a través de las partes inseguras.

HTTPS es un poco menos eficiente que HTTP por lo que si se utiliza en sitios donde la transferencia de información es muy grande puede notarse en el rendimiento. Por supuesto en el caso de que la información trasferida sea sensible, las ventajas compensan con creces a los inconvenientes.

El protocolo HTTPS usa el **puerto 443** por defecto. Al igual que sucedía con HTTP y el puerto 80, en caso de que se utilice este puerto no es necesario que el cliente lo indique en la barra de direcciones del navegador.

Certificados Digitales

El protocolo HTTPS encripta la comunicación para que quien capture tramas de ella no pueda ver los contenidos. Los navegadores web actuales basan el uso de HTTPS en el conocimiento de Autoridades Certificadoras que emiten Certificados Digitales para asegurar que el servidor al que nos conectamos es quien dice ser. Estas Autoridades Certificadoras son o agentes dedicados a ello específicamente o empresas como Microsoft.

El uso de HTTPS se basa en la confianza que nos proporcionen las entidades que emiten los certificados. Cuando usamos un navegador web determinado, la empresa que lo ha desarrollado ya ha introducido en él determinadas Autoridades Certificadoras que considera de confianza. Estas entidades se pueden consultar en el propio navegador y podemos modificarlas. En la imagen puedes ver parte de la lista incluida en Firefox.

> Investiga en los diferentes navegadores que tengas instalados cómo se consulta la lista de Autoridades Certificadoras en las que se confía.
>
> En la captura de imagen se pueden ver otras pestañas para el administrador de certificados. ¿Para qué sirve cada una de ellas?

Existen muchos servidores que usan certificados no emitidos por estas autoridades. En esos casos depende del usuario el aceptarlos o no. El navegador muestra una advertencia cuando vamos a conectarnos y tendremos que decidir si queremos seguir o preferimos no conectarnos con el destino.

En la imagen podemos ver un ejemplo con varias partes:

1. Si no confiamos en el destino debemos pulsar el botón para salir.
2. Podemos obtener más información. En este caso vemos que la entidad es gob.es por lo que podemos decidir confiar en él.
3. Si queremos confiar debemos leer los riesgos que conlleva.
4. Podemos añadir una excepción de seguridad para confiar en este sitio a partir de ahora.

> ¿Qué es la firma digital y cuál es su relación con los certificados digitales?

Un **Certificado Digital** es un documento electrónico que enlaza una clave pública con una Firma Digital e información personal sobre la persona u organización que quiere usar la clave pública. Sirve para asegurar que esa clave pertenece a dicha persona u organización. La información personal que se adjunta suele ser el nombre, la dirección, el correo electrónico, etc. La Firma Digital suele ser la de una Entidad Certificadora reconocida para que los clientes puedan confiar en que la Clave Pública y la información personal corresponden a la misma persona/organización. Este tipo de certificados se conocen como **Certificados Raíz** ya que se encuentran en el nivel más alto de un árbol de certificados. El usuario debe fiarse de la entidad emisora del Certificado Raíz ya que es el que asegura la autenticidad de todos los certificados emitidos por ella. La lista de certificados que se incluyen por defecto en un navegador web corresponden a esta categoría y el usuario confía en los desarrolladores del navegador para que se aseguren de que esos certificados son de confianza.

¡Como se puede ver la seguridad en Internet se basa en muchos niveles de confianza! Nadie debe asustarse. Las autoridades que incluyen los principales navegadores están verificadas y con revisadas con regularidad.

En esta página de la Wikipedia se puede obtener mucha más información sobre los Certificados Digitales.

En esta imagen se puede ver el uso de certificados digitales.

Los **Servidores de Certificados** son los que se encargan de validar o certificar las claves.

En la actualidad se utilizan varios formatos de Certificados Digitales en Internet. El más extendido es el X.509

Lee esta página y comenta en clase los puntos que consideres más interesantes sobre la información obtenida.

En nuestro navegador podemos consultar los detalles de un certificado. Si nos ponemos sobre el certificado determinado y pulsamos “Ver” suele aparecer una opción “Detalles” en la que podemos consultar datos como el algoritmo y el valor de la firma.

Obtener un certificado digital

Para poder utilizar HTTPS en nuestro servidor es necesario disponer de un Certificado Digital. Podemos obtener un Certificado Digital de una Autoridad Certificadora (*certificate authority, CA*) o crear nuestra propia Autoridad Certificadora y generar nuestros Certificados Digitales.

En España, podemos obtener Certificados Digitales a través de la Fábrica Nacional de Moneda y Timbre o mediante empresas como Verisign.

Para obtener un Certificado de una Autoridad Certificadora generalmente hay que demostrar que somos quienes decimos ser (cómo lo establece cada autoridad) y generar una petición para nuestro servidor (*certificate signing request, CSR*) que se debe enviar a la autoridad. Cuando nuestra petición haya sido aceptada, ya podemos instalar el certificado en nuestro servidor.

SSL/TSL

Secure Socket Layer (SSL) fue desarrollado por Netscape en los años 90. Han existido tres versiones de SSL pero actualmente se utiliza la 3.0 aunque deberíamos dejar de hacerlo ya que es un protocolo obsoleto con vulnerabilidades conocidas. Ha derivado en otro protocolo *Transport Layer Security* (TLS) que también tiene tres versiones, la 1.0, la 1.1 y la 1.2. La mayoría de los navegadores actuales usan la versión 1.0

Como la hemos comentado con antelación, HTTPS es una implementación de HTTP sobre SSL o TSL en el servidor.

Este protocolo se coloca sobre la capa de transporte donde los dos protocolos más típicos son TCP y UDP.

Es un protocolo que se utiliza para asegurar **confidencialidad**, **autenticidad**, **integridad** y **no repudio** entre el cliente y el servidor.

> Investiga y comenta los cuatro conceptos del párrafo anterior.

Existen dos modos: Uno en el que solo el servidor demuestra su identidad y otro en el que tanto el cliente como el servidor usan Certificados Digitales.

Las aplicaciones del uso de SSL/TSL son múltiples e incluyen la creación de redes privadas virtuales (VPN), el uso en comercio electrónico y en el correo electrónico, etc.

HTTPS en Apache

Apache utiliza un módulo específico basado en un proyecto que se llama OpenSSL; a pesar del nombre implementa también TLS. Para utilizar HTTPS en Apache es necesario que el módulo *mod_ssl* esté activo.

```
sudo a2enmod ssl
service apache2 restart
```

Ahora el servidor debería estar escuchando tanto el puerto 80 (http) como en el 443 (https). Si miramos el archivo de configuración de puertos

```
gedit /etc/apache2/ports.conf
```

en el que vemos los diferentes archivos y que podríamos usar GNUTLS.

```
Listen 80

<IfModule ssl_module>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

veremos que si el módulo *mod_ssl* está activo se añade la orden

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    Listen 443
</IfModule>
```

Hemos visto que esta distribución de Apache viene con dos sitios por defecto. El que no hemos usado se llama *Default-SSL* precisamente. Si lo activamos tendríamos ya un sitio con dicha configuración que escucharía por conexión segura.

```
sudo a2ensite default-ssl
service apache2 reload
```

Ahora tendríamos disponible la posibilidad de conectarnos de forma segura a ambos servidores. Si no especificamos el protocolo o usamos http se conectará de la forma estándar. Sin embargo, si nos conectamos mediante https nos muestra una excepción de seguridad como la que vimos antes. Esto es debido a que al instalar Apache se crea un certificado autofirmado para el sitio por defecto.

Si abrimos el archivo de configuración del sitio con ssl podemos ver cómo está configurado.

```
        #   A self-signed (snakeoil) certificate can be created by installing
```

```
        #   the ssl-cert package. See
        #   /usr/share/doc/apache2.2-common/README.Debian.gz for more info.
        #   If both key and certificate are stored in the same file, only the
        #   SSLCertificateFile directive is needed.
        SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

Tras realizar los pasos anteriores, prueba a interceptar tramas de conexiones http y https. ¿Qué diferencias ves? Guarda en un archivo una trama de cada tipo para su discusión en clase.

Si te fijas, los dos sitios por defecto tienen configurado el mismo directorio como raíz para los documentos. Qué pasaría si cambias el directorio raíz de uno de los dos. Pruébalo.

Es una mala opción dejar abierta la posibilidad de acceder al mismo sitio mediante una conexión segura y una insegura. Por ello si habilitamos un sitio con HTTPS no deberíamos tener activo el equivalente con HTTP como sucede con los dos por defecto. **Lo correcto sería deshabilitar el default.**

Si deshabilitamos el sitio por defecto podemos ver que ahora, si no especificamos el protocolo, utiliza HTTPS por defecto.

```
sudo a2dissite default
service apache2 reload
```

Estudia el resto del archivo de configuración del sitio por defecto con ssl y explica qué hacen el resto de directivas. Los enlaces a continuación contienen toda la información necesaria.

https://httpd.apache.org/docs/2.4/en/ssl/

https://httpd.apache.org/docs/2.4/mod/mod_ssl.html

Creando un sitio virtual con HTTPS

Ahora que hemos probado el sitio seguro por defecto, vamos a crear uno desde cero. Voy a dejar habilitado el sitio por defecto para hacer un ejemplo con un sitio con HTTP y otro con HTTPS.

```
sudo a2dissite default-ssl
sudo a2ensite default
service apache2 reload
```

Como ya hemos visto antes, para poder usar SSL en Apache es necesario tener un certificado. El que se instala para el sitio de ejemplo ya no es válido y tenemos que conseguir uno. Podemos adquirir uno de una CA o crear uno autofirmado. Por motivos evidentes nosotros usaremos esta última opción.

Para obtener un certificado es necesario generar una clave privada y para ello necesitamos un nombre de dominio así que lo primero será **configurar el DNS** correctamente. En mi caso voy a llamarlo www.con-ssl.es y habrá que configurarlo como un host virtual por nombre.

Para generar la clave se utiliza el comando genrsa

```
openssl genrsa -out clavepru.key 2048
```

Se puede generar una contraseña para la clave, pero si se va a utilizar para crear un certificado no es buena idea porque cada vez que el servidor web necesite acceder a la clave habrá que introducir la contraseña. Si no nos importa introducir la clave cada vez que se reinicie el servidor, usaríamos la opción

```
openssl genrsa –des3 -out clavepru.key 2048
```

Actualmente se recomiendan longitudes mínimas de clave de 2048 bits.

Lo siguiente será generar una petición para nuestro certificado. Esta petición es la que deberíamos enviar a la CA para obtener un certificado firmado por ellos. Luego esperaríamos a que lo firmaran y nos enviaran el certificado para instalarlo. Nosotros usaremos uno autofirmado.

```
openssl req -new -key clavepru.key -out peticionpru.csr
```

Cuando ejecutamos este comando nos va pidiendo información de la empresa para la que sea el certificado. La vamos rellenando hasta terminar. Los campos que terminan en [ ] no son obligatorios. Si hubiéramos generado la clave con contraseña nos la pediría antes de rellenar la información.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SergioCuesta
Organizational Unit Name (eg, section) []:SC
Common Name (e.g. server FQDN or YOUR name) []:Sergio Cuesta
Email Address []:una@con-ssl.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Para obtener un certificado autofirmado usaremos el comando

```
openssl x509 -req -days 365 -in peticionpru.csr -signkey clavepru.key -out certificadopru.crt
```

Lo que indica que usará el formato X.509 y tendrá una validez de un año. Si todo ha ido bien deberíamos ver algo como lo siguiente

```
Signature ok
subject=/C=ES/ST=Madrid/L=Madrid/O=SergioCuesta/OU=SC/CN=Sergio Cuesta/emailAddress=una@con-ssl.es
Getting Private key
```

En muchos sitios verás que en lugar de *crt* se crean archivos con la extensión *pem*. En teoría la diferencia es que *crt* solo contiene el certificado mientras que *pem* contiene tanto el certificado como la clave pero en la práctica esto se ignora y da igual usar una que otra.

Ten en cuenta que todos los archivos que se han creado en los pasos anteriores se han generado en el directorio en el que nos encontráramos por lo que hay que moverlos a los sitios adecuados. La petición no es necesaria.

```
sudo mv clavepru.key /etc/ssl/private/
sudo mv certificadopru.crt /etc/ssl/certs/
```

Crearemos un directorio para el contenido del sitio seguro

```
mkdir /var/www/con-ssl/
```

Procedemos a crear el sitio virtual por nombre

```
<IfModule mod_ssl.c>

NameVirtualHost 192.168.1.36:443

<VirtualHost 192.168.1.36:443>
    ServerName con-ssl.es
    ServerAlias www.con-ssl.es
    ServerAdmin alguien@con-ssl.es
    DocumentRoot /var/www/con-ssl

    <Directory /var/www/con-ssl>
        DirectoryIndex index.html
```

```
            Options -Indexes
            AllowOverride None
            Require all granted
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error_con_ssl.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/con_ssl_access.log combined

        # Esta es la parte de SSL
        SSLEngine on

        SSLCertificateFile /etc/ssl/certs/certificadopru.crt

        SSLCertificateKeyFile /etc/ssl/private/clavepru.key

        # Recuerda que lo siguiente es para mantener la compatibilidad con ciertas
        # versiones de Microsoft Internet Explorer
        BrowserMatch "MSIE [2-6]" \
            nokeepalive ssl-unclean-shutdown \
            downgrade-1.0 force-response-1.0

        BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
    </VirtualHost>

    </IfModule>
```

Habilitamos el sitio y recargamos Apache

```
sudo a2ensite con-ssl.es
```

```
service apache2 reload
```

Recuerda que el nombre que tenemos asociado en el DNS va a la dirección IP de la máquina así que **tendremos que usar** https://www.con.ssl.es para acceder al sitio seguro. Si usamos HTTP va al sitio por defecto. Para evitar esto deberíamos añadir otra entrada al DNS para que vaya al sitio sin SSL (puerto 80) y modificar la configuración del host virtual para que responda a las peticiones a ese otro nombre de dominio en lugar de a todo (*)

Si los hemos adquirido los certificados a través de una CA de confianza dejaría de aparecer el aviso cuando un cliente se conecta. Evidentemente para un sitio profesional es más que recomendable.

Para casos en los que estemos involucrados en el despliegue en una empresa muy grande o por ejemplo una universidad, puede interesarnos crear **nuestra propia CA** para uso propio. También es posible crear un sitio en el que **los clientes tengan que acceder mediante un certificado propio** como por ejemplo en la Agencia Tributaria. Ambas cosas se pueden hacer usando *OpenSSL* pero escapa totalmente al contenido del curso.

Desde el descubrimiento del **ataque Poodle** se ha hecho más importante configurar correctamente la versión que usemos de encriptación.

> Crea dos sitios diferentes, uno con HTTPS y el otro sin él. Configura todo correctamente para que al ir a un sitio o a otro use el protocolo adecuado sin necesidad de que el usuario lo especifique en la barra de direcciones del navegador web.

Despliegue de aplicaciones sobre servidores web y Empaquetado de aplicaciones web.

Estos dos conceptos no los vamos a estudiar aquí. Las aplicaciones en servidores web suelen utilizar al menos una base de datos y las que se despliegan en Apache se están utilizando constantemente en el módulo de “Desarrollo Web en Entorno Servidor”. El empaquetado de aplicaciones es un concepto que está más relacionado con Tomcat por lo que lo veremos en el tema siguiente.

En los casos en los que no instalemos nuestro propio servidor sino que contratemos un hosting, debemos considerar que tenga las características que necesitemos (por ejemplo que permita o proporcione el uso de MySQL y PHP). Generalmente accedemos a nuestra estructura de carpetas mediante FTP y a la base de datos mediante algún cliente.