El Protocolo HTTP

El Protocolo de Transferencia de HiperTexto *(Hypertext Transfer Protocol)* es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

Desde el punto de vista de las comunicaciones, está soportado sobre los servicios de conexión TCP/IP, y funciona de la misma forma que el resto de los servicios comunes de los entornos UNIX: un proceso servidor escucha en un puerto de comunicaciones TCP (por defecto, el 80), y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un **objeto o recurso** sobre el que actúan; cada objeto Web (documento HTML, fichero multimedia o aplicación CGI) es conocido por su URL.

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME. De esta forma, el protocolo puede intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, byte a byte, y la identificación MIME permitirá que el receptor trate adecuadamente los datos.

Las principales características del protocolo HTTP son:

- **Toda la comunicación entre los clientes y servidores se realiza a partir de caracteres de 8 bits**. De esta forma, se puede transmitir cualquier tipo de documento: texto, binario, etc., respetando su formato original.
- **Permite la transferencia de objetos multimedia.** El contenido de cada objeto intercambiado está identificado por su clasificación MIME.
- Existen tres verbos básicos (hay más, pero por lo general no se utilizan) que un cliente puede utilizar para dialogar con el servidor: **GET**, para recoger un objeto, **POST**, para enviar información al servidor y **HEAD**, para solicitar las características de un objeto (por ejemplo, la fecha de modificación de un documento HTML).
- **Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma**. Es decir, en una operación se puede recoger un único objeto.
- **No mantiene estado**. Cada petición de un cliente a un servidor no es influida por las transacciones anteriores. El servidor trata cada petición como una operación totalmente independiente del resto.
- **Cada objeto al que se aplican los verbos del protocolo está identificado a través de la información de situación del final de la URL**.

HTTP se diseñó específicamente para el World Wide Web: es un protocolo rápido y sencillo que permite la transferencia de múltiples tipos de información de forma eficiente y rápida. Se puede comparar, por ejemplo, con FTP, que es también un protocolo de transferencia de ficheros, pero tiene un conjunto muy amplio de comandos, y no se integra demasiado bien en las transferencias multimedia.

Etapas de una transacción HTTP

Para profundizar más en el funcionamiento de HTTP, veremos primero un caso particular de una transacción HTTP; en los siguientes apartados se analizarán las diferentes partes de este proceso.

Cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:

1. Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo *Location* del cliente Web.
2. El cliente Web descodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional (el valor por defecto es 80) y el objeto requerido del servidor.
3. Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.
4. Se realiza la petición. Para ello, se envía el comando necesario (GET, POST, HEAD,…), la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada (casi siempre HTTP/1.0) y un conjunto variable de información, que incluye datos sobre las capacidades del *browser*, datos opcionales para el servidor,…
5. El servidor devuelve la respuesta al cliente. Consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.
6. Se cierra la conexión TCP.

Este proceso se repite en cada acceso al servidor HTTP. Por ejemplo, si se recoge un documento HTML en cuyo interior están insertadas cuatro imágenes, el proceso anterior se repite cinco veces, una para el documento HTML y cuatro para las imágenes.

En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un cierto periodo de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado ***HTTP Keep Alive***, es empleado por la mayoría de los clientes y servidores modernos. Esta mejora es imprescindible en una Internet saturada, en la que el establecimiento de cada nueva conexión es un proceso lento y costoso.

Veamos el proceso completo con un ejemplo:

1. Desde un cliente se solicita la URL **http://www.unican.es/invest/default.html**
2. Se abre una conexión TCP/IP con el puerto 80 del sistema **www.unican.es**
3. El cliente realiza la solicitud, enviando algo similar a esto:

```
GET /invest/default.html HTTP/1.0 Operación solicitada+objeto+versión de
HTTP
Accept: text/plain Lista de tipos MIME que acepta o entiende
```

```
Accept: text/html el cliente
Accept: audio/*
Accept: video/mpeg
.. .. ..
Accept: */* Indica que acepta otros posibles tipos MIME
User-Agent: Mozilla/3.0 (WinNT; I) Información sobre el tipo de cliente
Línea en blanco, indica el final de la petición
```

4. El servidor responde con la siguiente información:

```
HTTP/1.0 200 OK Status de la operación; en este caso, correcto
Date: Monday, 7-Oct-96 18:00:00 Fecha de la operación
Server: NCSA 1.4 Tipo y versión del servidor
MIME-version: 1.0 Versión de MIME que maneja
Content-type: text/html Definición MIME del tipo de datos a devolver
Content-length: 254 Longitud de los datos que siguen
Last-modified: 6-Oct-96 12:30:00 Fecha de modificación de los datos
Línea en blanco
<HTML> Comienzo de los datos
<HEAD><TITLE>Recursos de investigación en UNICAN</TITLE></HEAD>
<BODY>
.. .. ..
.. .. ..
</HTML>
```

5. Se cierra la conexión.

Estructura de los mensajes HTTP

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta. La estructura general de los dos tipos de mensajes se puede ver en el siguiente esquema:

Mensaje de solicitud		**Mensaje de respuesta**
Comando HTTP + parámetros Cabeceras del requerimiento (línea en blanco) Información opcional		Resultado de la solicitud Cabeceras de la respuesta (línea en blanco) Información opcional

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor HTTP, mientras que en la respuesta contiene el resultado de la operación, un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (unas obligatorias y otras opcionales), que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par CR-LF (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML que devuelve un servidor o el contenido de un formulario que envía un cliente.

Los siguientes apartados describen con más detalle el contenido de cada una de las secciones de los mensajes. El conocimiento y empleo de los mensajes HTTP es necesario en las siguientes situaciones:

- Para diseñar módulos CGI, ya que es preciso construir una respuesta similar a la que el servidor HTTP proporciona al cliente.
- Para diseñar aplicaciones independientes que soliciten información a un servidor (automatizar la recuperación de documentos o construir robots de búsqueda) se debe construir una cabecera HTTP con la información de la petición al servidor.

Comandos del protocolo

Los comandos o verbos de HTTP representan las diferentes operaciones que se pueden solicitar a un servidor HTTP. El formato general de un comando es:

Nombre del comando	Objeto sobre el que se aplica	Versión de HTTP utilizada

Cada comando actúa sobre un objeto del servidor, normalmente un fichero o aplicación, que se toma de la URL de activación. La última parte de esta URL, que representa la dirección de un objeto dentro de un servidor HTTP, es el parámetro sobre el que se aplica el comando. Se compone de una serie de nombres de directorios y ficheros, además de parámetros opcionales para las aplicaciones CGI (ver *Ejecución de programas en un servidor HTTP* en la página *).

El estándar HTTP/1.0 recoge únicamente tres comandos, que representan las operaciones de recepción y envío de información y chequeo de estado:

- **GET** Se utiliza para recoger cualquier tipo de información del servidor. Se utiliza siempre que se pulsa sobre un enlace o se teclea directamente a una URL. Como resultado, el servidor HTTP envía el documento correspondiente a la URL seleccionada, o bien activa un módulo CGI, que generará a su vez la información de retorno.
- **HEAD** Solicita información sobre un objeto (fichero): tamaño, tipo, fecha de modificación… Es utilizado por los gestores de cachés de páginas o los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un fichero.
- **POST** Sirve para enviar información al servidor, por ejemplo los datos contenidos en un formulario. El servidor pasará esta información a un proceso encargado de su tratamiento (generalmente una aplicación CGI). La operación que se realiza con la información proporcionada depende de la URL utilizada. Se utiliza, sobre todo, en los formularios.

Un cliente Web selecciona automáticamente los comandos HTTP necesarios para recoger la información requerida por el usuario. Así, ante la activación de un enlace, siempre se ejecuta una operación GET para recoger el documento correspondiente. El envío del contenido de un formulario utiliza GET o POST, en función del atributo de <FORM METHOD="...">. Además, si el cliente Web tiene un caché de páginas recientemente visitadas, puede utilizar HEAD para comprobar la última fecha de modificación de un fichero, antes de traer una nueva copia del mismo.

Posteriormente se han definido algunos comandos adicionales, que sólo están disponibles en determinadas versiones de servidores HTTP, con motivos eminentemente experimentales. La última versión de HTTP, denominada 1.1, recoge estas y otras novedades, que se pueden utilizar, por ejemplo, para editar las páginas de un servidor Web trabajando en remoto.

- **PUT** Actualiza información sobre un objeto del servidor. Es similar a POST, pero en este caso, la información enviada al servidor debe ser almacenada en la URL que acompaña al comando. Así se puede actualizar el contenido de un documento.
- **DELETE** Elimina el documento especificado del servidor.
- **LINK** Crea una relación entre documentos.
- **UNLINK** Elimina una relación existente entre documentos del servidor.

Las cabeceras

Son un conjunto de variables que se incluyen en los mensajes HTTP, para modificar su comportamiento o incluir información de interés. En función de su nombre, pueden aparecer en los requerimientos de un cliente, en las respuestas del servidor o en ambos tipos de mensajes. El formato general de una cabecera es:

Nombre de la variable : Cadena ASCII con su valor

Los nombres de variables se pueden escribir con cualquier combinación de mayúsculas y minúsculas. Además, se debe incluir un espacio en blanco entre el signo **:** y su valor. En caso de que el valor de una variable ocupe varias líneas, éstas deberán comenzar, al menos, con un espacio en blanco o un tabulador.

Cabeceras comunes para peticiones y respuestas

- **Content-Type**: descripción MIME de la información contenida en este mensaje. Es la referencia que utilizan las aplicaciones Web para dar el correcto tratamiento a los datos que reciben.
- **Content-Length**: longitud en bytes de los datos enviados, expresado en base decimal.
- **Content-Encoding**: formato de codificación de los datos enviados en este mensaje. Sirve, por ejemplo, para enviar datos comprimidos (x-gzip o x-compress) o encriptados.
- **Date**: fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: Sunday, 12-Dec-96 12:21:22 GMT+01. No existe un formato único en las fechas; incluso es posible encontrar casos en los que no se dispone de la zona horaria correspondiente, con los problemas de sincronización que esto produce. Los formatos de fecha a emplear están recogidos en los RFC 1036 y 1123.
- **Pragma**: permite incluir información variada relacionada con el protocolo HTTP en el requerimiento o respuesta que se está realizando. Por ejemplo, un cliente envía un Pragma: no-cache para informar de que desea una copia nueva del recurso especificado.

Cabeceras sólo para peticiones del cliente

- **Accept**: campo opcional que contiene una lista de tipos MIME aceptados por el cliente. Se pueden utilizar * para indicar rangos de tipos de datos; tipo/* indica todos los subtipos de un determinado medio, mientras que */* representa a cualquier tipo de dato disponible.
- **Authorization**: clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado. La información incluye el formato de autorización empleado, seguido de la clave de acceso propiamente dicha. La explicación se incluye más adelante.
- From: campo opcional que contiene la dirección de correo electrónico del usuario del cliente Web que realiza el acceso.
- **If-Modified-Since**: permite realizar operaciones GET condicionales, en función de si la fecha de modificación del objeto requerido es anterior o posterior a la fecha proporcionada. Puede ser utilizada por los sistemas de almacenamiento temporal de páginas. Es equivalente a realizar un HEAD seguido de un GET normal.
- **Referer**: contiene la URL del documento desde donde se ha activado este enlace. De esta forma, un servidor puede informar al creador de ese documento de cambios o actualizaciones en los enlaces que contiene. No todos los clientes lo envían.
- **User-agent**: cadena que identifica el tipo y versión del cliente que realiza la petición. Por ejemplo, los *browsers* de Netscape envían cadenas del tipo User-Agent: Mozilla/3.0 (WinNT; I)

Cabeceras sólo para respuestas del servidor HTTP

- **Allow**: informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere esta respuesta. Por ejemplo, Allow: GET, POST.
- **Expires**: fecha de expiración del objeto enviado. Los sistemas de cache deben descartar las posibles copias del objeto pasada esta fecha. Por ejemplo, Expires: Thu, 12 Jan 97 00:00:00 GMT+1. No todos los sistemas lo envían. Puede cambiarse utilizando un <META EXPIRES> en el encabezado de cada documento.
- **Last-modified**: fecha local de modificación del objeto devuelto. Se puede corresponder con la fecha de modificación de un fichero en disco, o, para información generada dinámicamente desde una base de datos, con la fecha de modificación del registro de datos correspondiente.
- **Location**: informa sobre la dirección exacta del recurso al que se ha accedido. Cuando el servidor proporciona un código de respuesta de la serie 3xx, este parámetro contiene la URL necesaria para accesos posteriores a este recurso.
- **Server**: cadena que identifica el tipo y versión del servidor HTTP. Por ejemplo, Server: NCSA 1.4.
- **WWW-Autenticate**: cuando se accede a un recurso protegido o de acceso restringido, el servidor devuelve un código de estado 401, y utiliza este campo para informar de los modelos de autentificación válidos para acceder a este recurso.

Códigos de estado del servidor

Ante cada transacción con un servidor HTTP, éste devuelve un código numérico que informa sobre el resultado de la operación, como primera línea del mensaje de respuesta. Estos códigos aparecen en algunos casos en la pantalla del cliente, cuando se produce un error. El formato de la línea de estado es:

Versión de protocolo HTTP utilizada	Código numérico de estado (tres dígitos)	Descripción del código numérico

Dependiendo del servidor, es posible que se proporcione un mensaje de error más elaborado, en forma de documento HTML, en el que se explican las causas del error y su posible solución.

Existen cinco categorías de mensajes de estado, organizadas por el primer dígito del código numérico de la respuesta:

- **1xx**: mensajes informativos. Por ahora (en HTTP/1.0) no se utilizan, y están reservados para un futuro uso.
- **2xx**: mensajes asociados con operaciones realizadas correctamente.
- **3xx**: mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.
- **4xx**: errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.
- **5xx**: errores del servidor, que no ha podido llevar a cabo una solicitud.

Los más comunes se recogen en la siguiente tabla:

Código	Comentario	Descripción
200	`OK`	Operación realizada satisfactoriamente.
201	`Created`	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. Este nuevo objeto ya está disponible. Puede ser utilizado en sistemas de edición de documentos.
202	`Accepted`	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. El nuevo objeto no está disponible por el momento. En el cuerpo de la respuesta se debe informar sobre la disponibilidad de la información.
204	`No Content`	La operación ha sido aceptada, pero no ha producido ningún resultado de interés. El cliente no deberá modificar el documento que está mostrando en este momento.
301	`Moved Permanently`	El objeto al que se accede ha sido movido a otro lugar de forma permanente. El servidor proporciona, además, la nueva URL en la variable Location de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. En caso de tener capacidad, el

		cliente puede actualizar la URL incorrecta, por ejemplo, en la agenda de *bookmarks*.
302	`Moved Temporarily`	El objeto al que se accede ha sido movido a otro lugar de forma temporal. El servidor proporciona, además, la nueva URL en la variable Location de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. El cliente no debe modificar ninguna de las referencias a la URL errónea.
304	`Not Modified`	Cuando se hace un GET condicional, y el documento no ha sido modificado, se devuelve este código de estado.
400	`Bad Request`	La petición tiene un error de sintaxis y no es entendida por el servidor.
401	`Unauthorized`	La petición requiere una autorización especial, que normalmente consiste en un nombre y clave que el servidor verificará. El campo WWW-Autenticate informa de los protocolos de autentificación aceptados para este recurso.
403	`Forbidden`	Está prohibido el acceso a este recurso. No es posible utilizar una clave para modificar la protección.
404	`Not Found`	La URL solicitada no existe.
500	`Internal Server Error`	El servidor ha tenido un error interno, y no puede continuar con el procesamiento.
501	`Not Implemented`	El servidor no tiene capacidad, por su diseño interno, para llevar a cabo el requerimiento del cliente.
502	`Bad Gateway`	El servidor, que está actuando como proxy o pasarela, ha encontrado un error al acceder al recurso que había solicitado el cliente.
503	`Service Unavailable`	El servidor está actualmente deshabilitado, y no es capaz de atender el requerimiento.

Cachés de páginas y servidores proxy

Muchos clientes Web utilizan un sistema para reducir el número de accesos y transferencias de información a través de Internet, y así agilizar la presentación de documentos previamente visitados. Para ello, almacenan en el disco del cliente una copia de las últimas páginas a las que se ha accedido. Este mecanismo, denominado **"caché de páginas"**, mantiene la fecha de acceso a un documento y comprueba, a través de un comando HEAD, la fecha actual de modificación del mismo. En caso de que se detecte un cambio o actualización, el cliente accederá, ahora a través de un GET, a recoger la nueva versión del fichero. En caso contrario, se procederá a utilizar la copia local.

Las versiones de Netscape Navigator anteriores a la 2.02 tienen fallos en su gestión de caché, que hace que se muestre una copia antigua de un documento almacenada en un caché, a pesar de que el documento original haya cambiado y se solicite la nueva versión a través de un *RELOAD*.

Un sistema parecido, pero con más funciones es el denominado **"servidor *proxy*"**. Este tipo de servidor, una mezcla entre servidor HTTP y cliente Web, realiza las funciones de cache de páginas para un gran número de clientes. Todos los clientes conectados a un *proxy* dejan que éste sea el encargado de recoger las URLs solicitadas. De esta forma, en caso de que varios clientes accedan a la misma página, el servidor *proxy* la podrá proporcionar con un único acceso a la información original.

La principal ventaja de ambos sistemas es la drástica reducción de conexiones a Internet necesarias, en caso de que los clientes accedan a un conjunto similar de páginas, como suele ocurrir con mucha frecuencia. Además, determinadas organizaciones limitan, por motivos de seguridad, los accesos desde su organización al exterior y viceversa. Para ello, se dispone de sistemas denominados "cortafuegos" *(firewalls)*, que son los únicos habilitados para conectarse con el exterior. En este caso, el uso de un servidor *proxy* se vuelve indispensable.

En determinadas situaciones, el almacenamiento de páginas en un caché o en un *proxy* puede hacer que se mantengan copias no actualizadas de la información, como por ejemplo en el caso de trabajar con documentos generados dinámicamente. Para estas situaciones, los servidores HTTP pueden informar a los clientes de la expiración del documento, o de la imposibilidad de ser almacenado en un caché, utilizando la variable Expires en la respuesta del servidor.

Magic Cookies

Las '***galletas mágicas'*** son una de las incorporaciones más recientes al protocolo HTTP, y son parte del nuevo estándar HTTP/1.1. Las *cookies* son pequeños ficheros de texto que se intercambian los clientes y servidores HTPP, para solucionar una de las principales deficiencias del protocolo: la falta de información de estado entre dos transacciones. Fueron introducidas por Netscape, y ahora han sido estandarizadas en el RFC 2109.

Cada intercambio de información con un servidor es completamente independiente del resto, por lo que las aplicaciones CGI que necesitan recordar operaciones previas de un usuario (por ejemplo, los supermercados electrónicos) pueden optar por dos soluciones, que complican bastante su diseño: almacenar temporalmente en el sistema del servidor un registro de las operaciones realizadas, con una determinada política para eliminarlas cuando no son necesarias, o bien incluir esta información en el código HTML devuelto al cliente, como campos HIDDEN de un formulario.

Las *cookies* son una solución más flexible a este problema. La primera vez que un usuario accede a un determinado documento de un servidor, éste proporciona una *cookie* que contiene datos que relacionarán posteriores operaciones. El cliente almacena la *cookie* en su sistema para usarla después. En los futuros accesos a este servidor, el *browser* podrá proporcionar la *cookie* original, que servirá de nexo entre este acceso y los anteriores. Todo este proceso se realiza automáticamente, sin intervención del usuario. El siguiente ejemplo muestra una *cookie* generada por la revista *Rolling Stone*.

```
EGSOFT_ID
191.46.211.13-655193640.29148285
www.rollingstone.com/
0
2867435528
30124157
```

La información dentro de una *cookie* se organiza a partir de líneas de texto. El campo más interesante es la tercera línea. El cliente Web la compara cada URL a la que accede; si se produce una concordancia entre ambas, se enviará la *cookie* correspondiente. Es decir, una *cookie* asociada a www.rollingstone.com/shop sólo se enviará con accesos por debajo de la sección /shop, mientras que la *cookie* del ejemplo servirá para todo el servidor *Rolling Stone*. Las *cookies* pueden tener asociada una fecha de expiración, a partir de la cual el cliente Web tratará de obtener una nueva versión.

¿Para qué se pueden utilizar? Su aplicación más inmediata son los sistemas de compra electrónica. Estos supermercados virtuales necesitan relacionar el contenido de un pedido con el cliente que lo ha solicitado. Sin *cookies*, la solución más sencilla es incluir dentro de los documentos HTML el contenido de las operaciones o pedidos previos, a través de variables ocultas. Con ellas es posible relacionar los sucesivos accesos al sistema con su origen y simplificar la gestión de la aplicación Web.

Otro uso muy interesante son los sistemas personalizados de recepción de información, en los que es posible construir una página a medida, con información procedente de fuentes muy diversas (ver por ejemplo my.yahoo.com/ o www.snap.com). En el primer acceso al sistema, se procede al registro; a partir de él, se genera una *cookie* que recibe el cliente. En accesos sucesivos, el cliente enviará la *cookie*, y el servidor podrá generar una página personalizada con las preferencias del usuario.

Por último, algunas compañías emplean las *cookies* para realizar un seguimiento de los accesos a sus servidores WWW, identificando las páginas más visitadas, la manera en que se pasa de una a otra sección, etc.

Para utilizar las *cookies*, es necesario que los clientes estén preparados para recogerlas y almacenarlas. Netscape Navigator e Internet Explorer disponen ya de esta capacidad. Sin embargo, los servidores no necesitan capacidades especiales, ya que son las aplicaciones CGI las encargadas de su gestión.

Por defecto, los clientes Web reciben y envían *cookies* sin solicitar confirmación al usuario, pero es posible recibir avisos de la recepción de *cookies*, para rechazarlas en caso de no ser de nuestro agrado. ¿Por qué rechazar las *cookies*?. Bueno, se pueden utilizar para seguir una pista del tipo de información que buscamos en Internet, y ofrecer información comercial en función de ello. Sin embargo, algunos servicios Web como tiendas on-line dependen de ellas para operar correctamente.

Las *cookies* tienen un tiempo de vida, que hace que sean descartadas pasado un cierto tiempo de actividad. Algunas expiran al salir del cliente Web, pero otras *cookies* tienen una duración mayor, por lo que el cliente Web las almacena en un fichero. Netscape utiliza el fichero cookies.txt de su directorio de instalación, mientras que Microsoft almacena cada uno en un fichero independiente del caché de páginas.

Las *cookies* son imprescindibles para operar con determinados servidores de acceso restringido, en los cuales es preciso seguir un proceso de registro, y posiblemente abonar algún tipo de tasa. Cuando se pierde una de esas *cookies*, por una reinstalación, fallo del ordenador o limpieza del caché de páginas, en algunos casos puede ser posible regenerarla siguiendo un nuevo proceso de registro en el servidor que la proporcionó, de forma que se relacione algún tipo de información personal o clave de acceso proporcionado en la primera visita al servidor.

Uso de las *cookies*

Una ***cookie*** es simplemente una serie de líneas de texto, con pares variable/valor. Existe un conjunto predefinido de nombres de variable, necesarias para el correcto funcionamiento de las *cookies*, pero se pueden crear nuevas variables para cubrir las necesidades de una aplicación concreta. Las principales variables predefinidas son:

- **Domain=** conjunto de direcciones Internet para el que es válida la *cookie*. Se puede dar una dirección única (www.mitienda.es) o un rango (.netscape.com).
- **Path=** fija el subconjunto de URLs para las que sirve esta *cookie*.
- **Version=** Permite seleccionar entre diferentes versiones del modelo de *cookies*.
- **Expires=** Fecha de expiración de la información. Si no se incluye, los datos son descartados al finalizar la sesión con el cliente Web. En caso contrario se almacenará en el disco del cliente. El RFC 2109 ha cambiado esta variable por Max-Age, una duración relativa en segundos.

Un servidor HTTP envía los diferentes campos de una *cookie* con la nueva cabecera HTTP Set-Cookie:

```
Set-Cookie: Domain=www.unican.es; Path=/; Nombre=Luis; Expires Fri,
15-Jul-97 12:00:00 GMT
```

Cuando se accede a una URL que verifica el par dominio/path registrado, el cliente enviará automáticamente la información de los diferentes campos de la cookie con la nueva cabecera HTTP Cookie:

```
Cookie: Domain=www.unican.es; Path=/; Nombre=Luis
```

Se puede encontrar más información sobre cookies y otros temas relacionados en http://www.w3.org/pub/WWW/Protocols/ y http://www.cookiecentral.com/