Servicios de transferencia de hipertexto HTTP


```
Servicios en red
CFGM Sistemas microinformáticos y redes (SMX)
IES Francesc de Borja Moll
Daniel Moreno <danimrprofe@gmail.com>
```


La World Wide Web

Una página web consiste en un *archivo HTML base* _ _ que incluye referencias a varios objetos

Estos *objetos* pueden ser archivos HTML, imágenes JPEG, Applets JAVA, archivos de AUDIO, scripts, CSS, etc.

Cada objeto se puede direccionar mediante una *URL*

Los contenidos de las páginas web pueden ser de diferentes tipos: textos, elementos multimedia, como imágenes, vídeos, etc.

Su reproducción o visualización requiere de otros **complementos** que son añadidos al navegador y que quedan integrados en él de forma transparente al usuario.

- Es posible ejecutar determinadas aplicaciones complementarias que aumentan la funcionalidad de la página web.
- **Aplicaciones que se ** *ejecutan en el cliente web*

    - Se ejecutan directamente en el equipo del usuario.
    - El servidor envía el código al navegador (suele ser en Java o JavaScript) y este lo ejecuta.
    - Lógicamente, el navegador debe ser capaz de ejecutarlo, y para ello a menudo requiere de la instalación de extensiones.
- **Aplicaciones que se ** _ejecutan en el servidor _
    - Suelen ser en PHP, Perl, Python, etc.
    - Al ejecutarse estos programas generan un código HTML que es enviado al navegador. Este lo interpreta y lo muestra al usuario.

Estructura de la WWW

La *WWW* es un entramado de documentos escrito en un formato de texto llamado hipertexto

El *hipertexto* es un sistema de organización de la información basado en la posibilidad de moverse por dentro de un texto y hacia otros textos diferentes mediante palabras clave

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Example<title>
        <link rel="stylesheet" href="styl
    </head>
    <body>
        <h1>
            <a href="/">Header</a>
        </h1>
        <nav>
            <a href="one/">One</a>
            <a href="two/">Two</a>
            <a href="three/">Three</a>
        </nav>
```

El lenguaje basado en hipertexto más conocido es el lenguaje *HTML* (Hyper-text markup language). La versión más actual de HTML es la 5

HTML Versions

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML 5	2014

Comparativa de versiones

Item	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Hyperlinks	x	x	x	x	x
Images	x	x	x	x	x
Lists	x	x	x	x	x
Active maps & images		x	x	x	x
Forms		x	x	x	x
Equations			x	x	x
Toolbars			x	x	x
Tables			x	x	x
Accessibility features				x	x
Object embedding				x	x
Style sheets				x	x
Scripting				x	x
Video and audio					x
Inline vector graphics					x
XML representation					x
Background threads					x
Browser storage					x
Drawing canvas					x

Estructura de la WWW

- La estructura del hipertexto está formada por 3 elementos esenciales
  - *Nodos* : unidades básicas que contienen información
  - *Enlaces* : interconectan los nodos vinculando segmentos de información
  - *Anclajes* : se utilizan para marcar el inicio y destino de cada enlace
- Todos estos elementos permiten acceder a la información a través de la *navegación* entre diferentes páginas

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

Direcciones URL

El localizador uniforme de recursos o *URL* es una dirección que permite encontrar y acceder a un objeto concreto en internet.

Está formada por una secuencia de caracteres que sigue una determinada *estructura*

No es obligatorio especificar todos los componentes de la URL. En caso de no hacerlo, el navegador presupone un *valor por defecto* .

Por ejemplo, si no especificamos protocolo se entenderá que es HTTP. Si no especificamos un puerto, será en puerto 80.

HTTP

- La URL es una cadena de texto formada por:
  - *Protocolo* de comunicaciones (**http, ftp, https...) ** – HTTP en muchos casos
  - _Host o IP _ (**www.softuni.bg, gmail.com, 127.0.0.1, web**)
  - *Puerto* al que queremos conectar. El puerto por defecto es 80, pero podría ser uno del rango [0...65535]
  - *Ruta* (**/ forum , / path / index.php**)
  - *Cadena query* (**?id=27&lang=en**)
  - *Fragmento* (**#lectures**) – usado en el cliente para navegar a alguna sección

URL válidas y no válidas

- Las URL se codifican acorde a la norma RFC 1738 (https://www.ietf.org/rfc/rfc1738.txt)
- Únicamente se pueden utilizar los siguientes caracteres dentro de la URL

```
[0-9a-zA-Z], $, -, _, . , +, *, ', (, ), ,, !
```

- Si queremos utilizar en la URL un carácter no válido, se pueden indicar codificados precedidos por **%+ un código hexadecimal**
- No están permitidos los espacios en una URL, para indicarlo utilizaríamos el carácter %20

Char	URL Encoding
space	%20
щ	%D1%89
"	%22
#	%23
$	%24
%	%25
&	%26

URL según diferentes protocolos

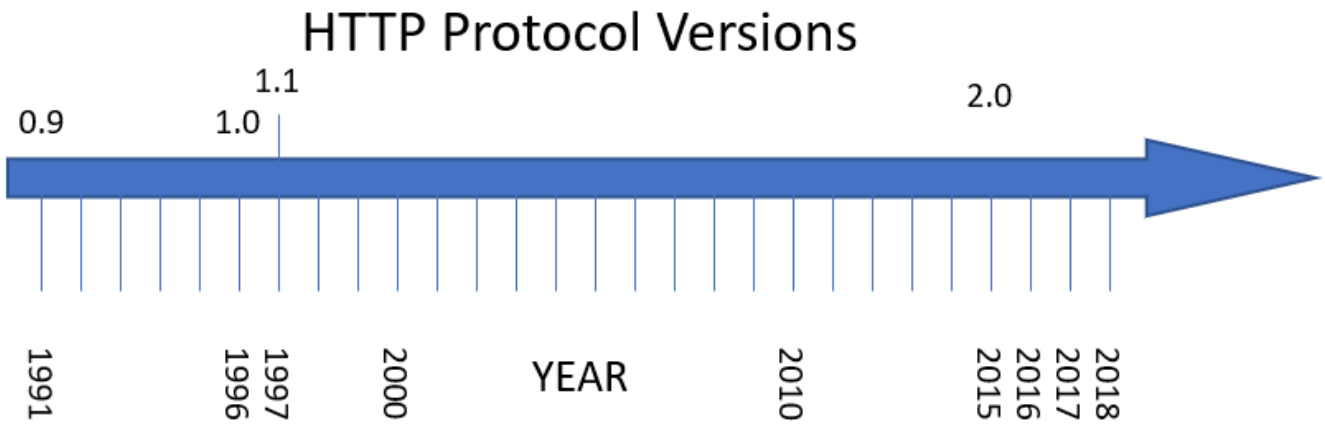
Name	Used for	Example
http	Hypertext (HTML)	http://www.ee.uwa.edu/~rob/
https	Hypertext with security	https://www.bank.com/accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
mailto	Sending email	mailto:JohnUser@acm.org
rtsp	Streaming media	rtsp://youtube.com/montypython.mpg
sip	Multimedia calls	sip:eve@adversary.com
about	Browser information	about:plugins

Protocolo HTTP

El protocolo de transferencia de hipertexto o HTTP establece el protocolo para el intercambio de documentos de hipertexto y contenido multimedia en Internet

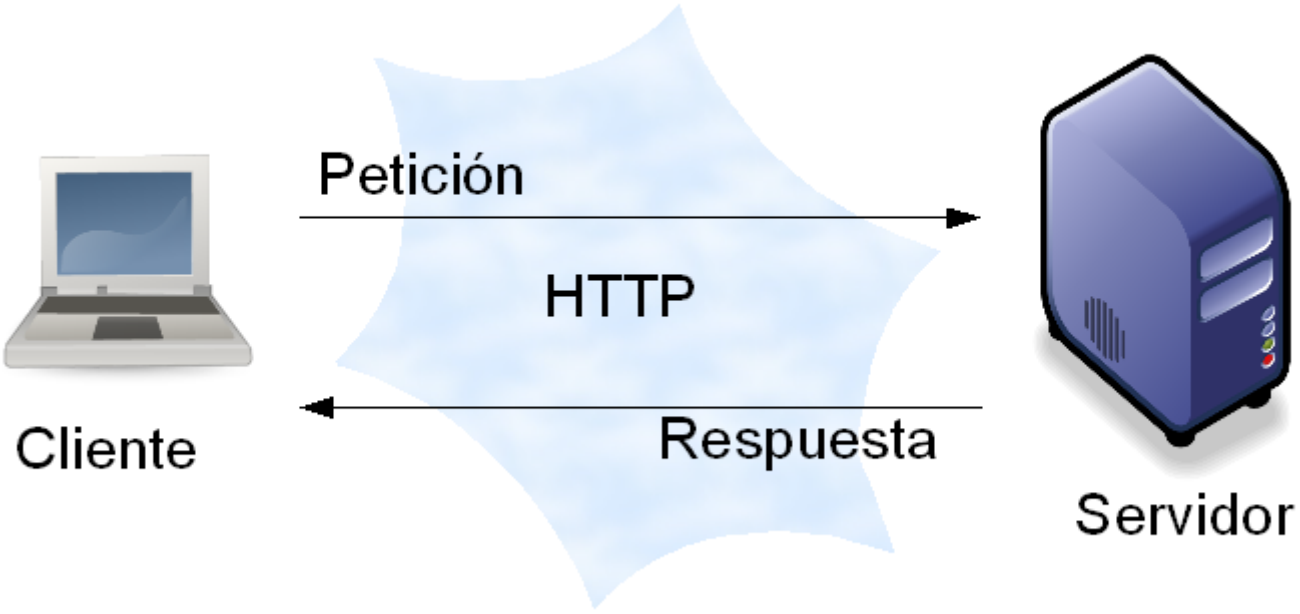
HTTP fue desarrollado por la W3C y la IETF en 1999 a través de la especificación RFC 2616

Las *versiones* de HTTP son HTTP/1.0, 1.1, 1.2 y 2



Clientes y servidores

- _Clientes web: _
  - Máquinas que acceden a la información en la web
  - Los clientes se conectan a la www utilizando un navegador o web browser, un software utilizado por el usuario final para acceder a la web
- _Servidor web: _
  - Proporciona contenido web a través del protocolo HTTP



¿Qué es una página web?

- Una _página web _ (web page) consiste en un archivo HTML base que incluye referencias a un conjunto de objetos, que pueden ser:
  - Páginas HTML, Imágenes JPEG
  - Applets Java, Archivos de Audio
  - Scripts (JavaScript), Hojas de estilos (CSS)
- Estos objetos están localizados en diferentes servidores de internet. Una imagen o un vídeo de la página puede estar alojada en otro servidor.
- Cada objeto se puede direccionar mediante una dirección URL diferente, que indica la ruta para llegar a él.

HTTP

- **Las principales características de HTTP son:**
- Utiliza una estructura _cliente/servidor. _ Los servidores alojan las páginas web, y los clientes acceden a ellas a través de un navegador.
- Para visualizar los datos a través de HTTP se requiere un *navegador* _ _ _web. _ El navegador analiza el contenido de las páginas, interpreta la forma de la página y la representa en pantalla.
- Las páginas se pueden ver *en cualquier dispositivo* , independientemente del hardware y SO que utilice.
- **Las principales características de HTTP son:**
  - Para la comunicación se establece una conexión TCP a través del _puerto 80 _ (por defecto). Todos los servidores web escuchan en el mismo puerto, y por ello no hace falta especificarlo cada vez que se visita una página.

- La comunicación se basa en mensajes de _petición y respuesta. _ Para cada página diferente a la que accedemos, se crea una petición. El servidor contesta a esta petición con el contenido de la página.
- Se crea una petición y respuesta por cada *objeto* que contiene la página.

Ejemplo de sesión

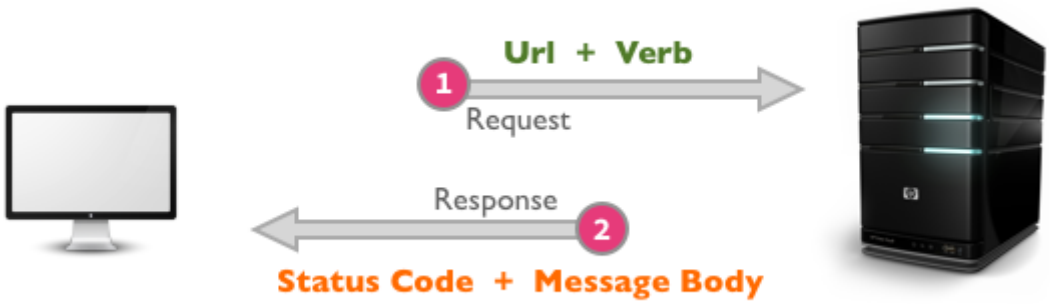
- Una *sesión* HTTP es una secuencia de transacciones solicitud y respuesta
- El *funcionamiento* básico de una sesión HTTP es:
  - El usuario escribe una dirección en el navegador (cliente)
  - El navegador realiza una consulta DNS para averiguar la dirección IP asociada a la URL
  - Intenta establecer una conexión TCP al puerto 80 (predeterminado para servidores web)
  - Cuando esta conexión se ha establecido, el navegador envía la petición HTTP solicitando la URL

Ejemplo de sesión

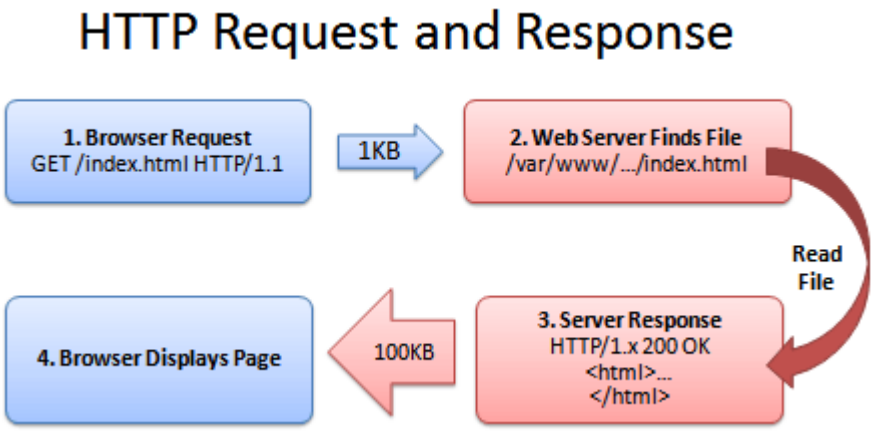
Una vez establecida la conexión HTTP, el cliente pide mediante una solicitud el recurso que necesita (html, css, javascript, etc)

El servidor contesta con un código de estado y el recurso que le ha pedido el cliente, si es que existe.

Todo este proceso es *transparente al usuario* , que únicamente ve la carga de la página. Todo lo realiza el navegador

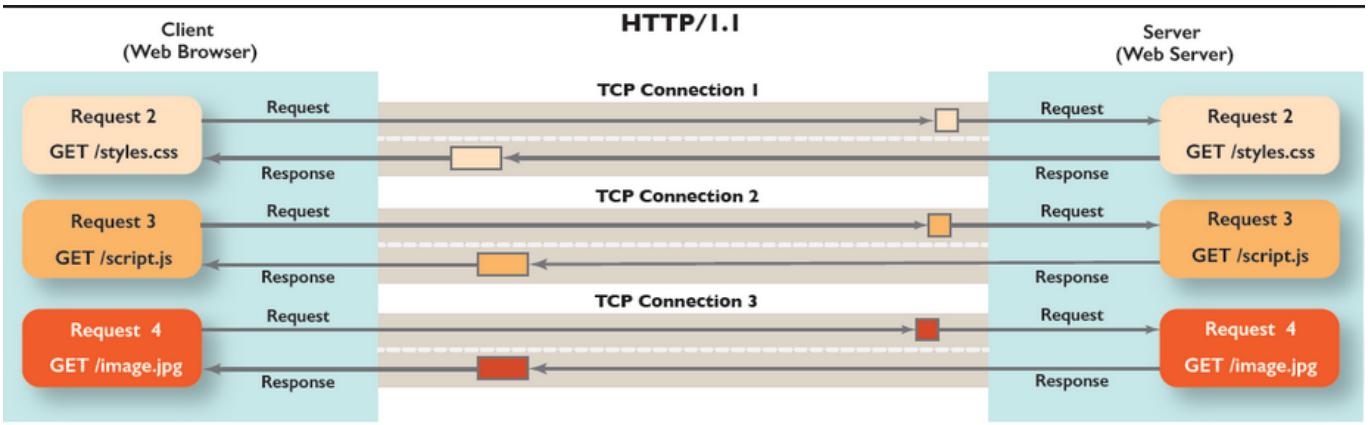


Ejemplo de sesión



Ejemplo de sesión

- Cuando una página web contiene imágenes, scripts, u otros objetos externos, el navegador los pide uno por uno
- En el ejemplo vemos que la página necesita:
  - Un _hoja de estilos _ CSS que definen la apariencia de la página
  - Un archivo de *script* JavaScript que contendrá funciones para hacer que la página sea dinámica.
  - Una *imagen* en formato png



Peticiones HTTP

Ejemplo de sesión

- El mensaje de *solicitud* del cliente tiene la siguiente estructura:
  - Línea de petición (request line)
  - Cabeceras de petición (request headers)
  - Una línea en blanco separa las cabeceras del cuerpo del mensaje
  - Un cuerpo de mensaje, si fuese necesario especificar algo más.

HTTP

Formato de la petición

- *Línea de petición*
- Es la primera línea de una petición. Por ejemplo:

```
GET  /Ciclos/CFGS/DAW.html   HTTP/1.1
```

- *Formato*
- Está formada por 3 partes separadas por espacios
- En primer lugar el **método** de petición HTTP: *GET*. Se trata por tanto de una petición GET (queremos que nos den algo)
- Identificador URL del recurso. Tenemos que especificar la ruta hacia el objeto que queremos: */Ciclos/CFGS/DAW.html. _ Si queremos el documento raíz, indicaremos únicamente _ /*
- Versión HTTP que queremos utilizar: *HTTP/1.1*

Métodos

- El método utilizado por defecto por los navegadores es el **GET** . Pero en determinadas situaciones se pueden utilizar otros.

- HTTP define 8 *métodos* , los cuales indican la acción que se desea realizar sobre el recurso

    - GET
    - HEAD
    - POST
    - PUT
    - DELETE
    - TRACE
    - OPTIONS
    - CONNECT

- *Método GET*

    - Solicita una representación del recurso solicitado
    - GET /images/logo.png HTTP/1.1 (obtiene una imagen)
    - GET /pages/index.html HTTP/1.1 (obtiene una página web)

- *M étodo* _ HEAD_

    - Solicita una respuesta, idéntica a la que se generaría en una consulta GET, pero sin el cuerpo de la respuesta
    - Es útil para conseguir los metadatos incluidos en la cabecera de la respuesta, sin tener que enviar todo el contenido

- *Método POST*

    - Envía datos para ser procesados (en un formulario HTML) a un recurso específico

- *Método PUT*

    - Carga en el servidor un recurso especificado (archivo)
        - PUT /path/filename.html HTTP/1.1

- *Método DELETE*

    - Borra un recurso especificado.

- *Método TRACE*
    - Solicita al servidor que en el mensaje de respuesta incluya el mensaje de solicitud
    - Se utiliza en tareas de diagnóstico y comprobación
- *Método OPTIONS*
    - Retorna los métodos HTTP para un recurso URL específico
    - Permite adivinar qué se puede hacer sobre un recurso URL
- *Método CONNECT*
    - Se utiliza para convertir la conexión HTTP en un túnel TCP/IP transparente para utilizar comunicaciones cifradas mediante el protocolo SSL

Cabeceras

- Las cabeceras indican información extra que queremos hacer llegar al cliente o al servidor, relacionada con la petición o la respuesta.
- Existen cabeceras que se pueden utilizar solo en una petición, en la respuesta o en ambas. Son opcionales y pueden o no incluirse en los mensajes.
- El formato de las cabeceras es *nombre:valor*
- **Ejemplos de cabeceras de petición**
    - _Host: _ especifica el recurso solicitado
    - *User-agent* _: _ informa del navegador que se utiliza
    - *Accept-language* _: _ indica en que idioma se desea recibir la página solicitada
    - *Accept-encoding* _, _ *accept-charset* _: _ indican al servidor que tipo de contenidos conoce el navegador y sabe representar

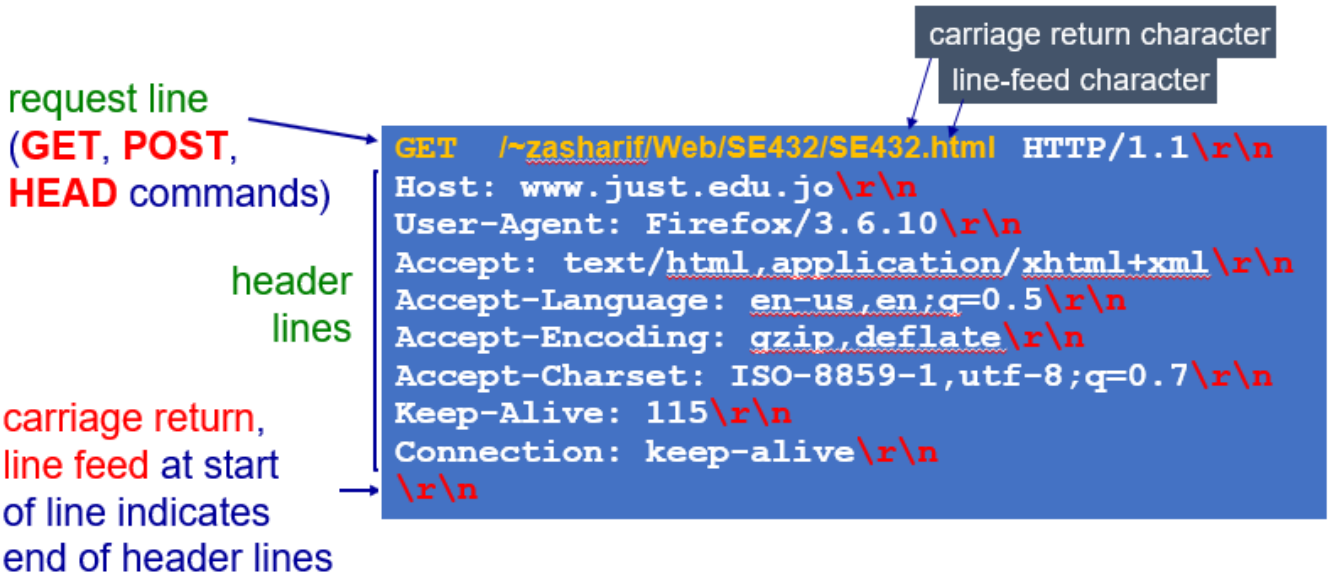
Ejemplo de sesión

```
GET / HTTP/1.1
Host: www.ies.edu
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
    rv:1.2b) Gecko/20021016
Accept:
    text/xml,application/xml,application/xhtml+xml,t
    ext/html;q=0.9,text/css,*/*;q=0.1
Accept-Language: es-es, en-us;q=0.66, en;q=0.33
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-15, utf-8;q=0.66,
    *;q=0.66
Keep-Alive: 300
Connection: keep-alive
```

Cabeceras de petición

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
If-Modified-Since	Request	Time and date to check freshness
If-None-Match	Request	Previously sent tags to check freshness
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Referer	Request	The previous URL from which the request came
Cookie	Request	Previously set cookie sent back to the server

Formato de petición HTTP



Respuestas HTTP

Protocolo HTTP

Ejemplo de respuesta

El servidor informa del éxito de la petición (200 OK) y de la versión del protocolo que conoce

Indica la fecha y hora del servidor, y versión y módulos del servidor web

Content-type: indica el tipo de contenido (text/html)

Por último, el contenido propiamente dicho del recurso solicitado (archivo index.html)

```
HTTP/1.1 200 OK
Date: Sun, 10 Nov 2002 22:50:55 GMT
Server: Apache/1.3.26 (Unix) mod_bwlimited/1.0
    PHP/4.2.2 mod_log_bytes/0.3
    FrontPage/5.0.2.2510 mod_ssl/2.8.9
    OpenSSL/0.9.6b
Content-Type: text/html
Age: 130
<html>
 <body>
    <h1>Página principal de tuHost</h1>
(Contingut) . . .
</body>
</html>
```

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Códigos de estado

El servidor responde con códigos de estado. El **código de estado ** es importante y le dice al cliente cómo interpretar la respuesta del servidor

Los códigos de estado están formados por 3 dígitos y incluyen una descripción. Ejemplo: **404 ** **Not** ** ** **Found** ** **

Los códigos de estado se dividen en 5 **clases** diferentes, según el primer dígito:

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

- *Los otros 2 dígitos proporcionan información adicional*
- **200 OK: ** Todo funcionó bien, aquí están los datos
- **301 Moved Permanently :** El objeto solicitado se ha movido, y se especifica la nueva localización (Location:)
- **302 Moved ** **temporarily** **: ** La URL está temporalmente fuera de servicio, utiliza esta que está en otro sitio
- **400 ** **Bad** ** ** **Request** **: ** Hay un error de sintaxis en la petición
- **403 ** **Forbidden** : No puedes hacer esto, y no te diremos porqué
- **404 ** **Not** ** ** **Found** **: ** No existe el documento
- **408 ** **Request ** Time- out** **: ** La petición tardó demasiado en llevarse a cabo por algún motivo
- **505 HTTP Version ** **Not** ** ** **Supported**

En este caso, el servidor contesta informando que el contenido pedido ha sido **movido de lugar ** para siempre.

También nos informa de la nueva **dirección** en la que podemos encontrarlo.

El navegador hace la redirección **automáticamente** , sin que el usuario intervenga.

```
GET / HTTP/1.1
Host: http://softuni.org
User-Agent: Gecko/20100115 Firefox/3.6
<CRLF>
```

Redirección

```
HTTP/1.1 301 Moved Permanently
Location: http://softuni.bg
```

Respuesta HTTP

Cabeceras de respuesta

Las puede incluir el servidor en sus mensajes de respuesta

Set-Cookie	Response	Cookie for the client to store
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., *gzip*)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Content-Range	Response	Identifies a portion of the page's content
Last-Modified	Response	Time and date the page was last changed
Expires	Response	Time and date when the page stops being valid
Location	Response	Tells the client where to send its request
Accept-Ranges	Response	Indicates the server will accept byte range requests

Cookies

- *Navegar* por la web significa realizar una serie de peticiones de páginas diferentes.
- El servidor *olvida* lo que ha sucedido anteriormente, no sabe lo que hemos hecho en páginas anteriores.
- No es adecuado cuando una página debe mostrar información diferente a cada usuario dependiendo de lo que han hecho anteriormente.
    - Páginas con subscripción
    - Comercio electrónico
- ¿Cómo podríamos *rastrear* a los usuarios?

¿Podríamos rastrear a los usuarios diferenciándolos por IP?

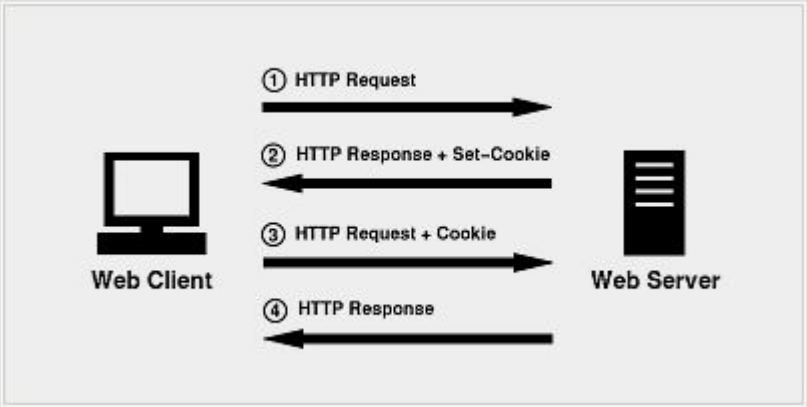
En las casas y oficinas, varios equipos comparten la misma IP pública. Estos equipos se ven desde fuera como uno solo.

Una IP identifica un ordenador, no el usuario que hay detrás. Varios usuarios pueden utilizar el mismo ordenador (misma IP)

Muchos ISP asignan IP a los clientes utilizando DHCP. La IP que tenemos en un momento dado, podría de repente ser la de tu vecino.

Cuando un cliente solicita una página web, el servidor le puede enviar _información adicional _ en forma de cookie, además de la página.

Una cookie es una _cadena de texto _ (máximo 4 KB) que el servidor puede asociar al navegador



Las cookies permiten que el sitio web **recuerde** las acciones y preferencias de un usuario

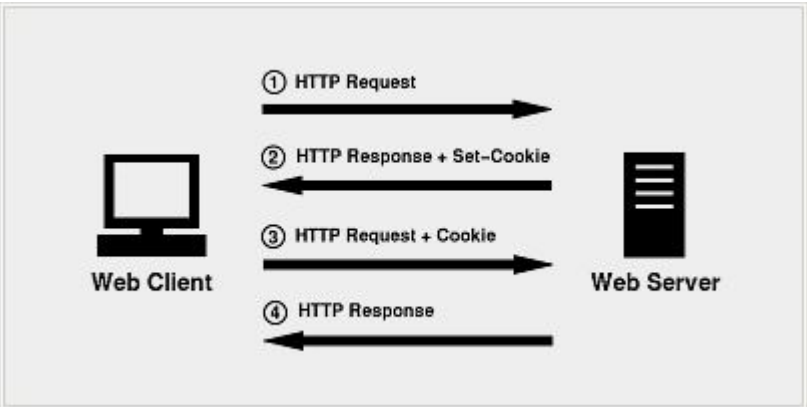
Por ejemplo se puede **guardar** el identificador de sesión, el idioma, el tamaño de letra, etc.

Cuando regrese al sitio o navegue por las páginas no tiene que volver a configurarlo

Antes de que el navegador envíe una petición para una página a un website, mira en la carpeta de cookies del disco.

**Comprueba si tiene cookies ** para el dominio al que va a hacer la solicitud. Si hay alguna, las incluye en el mensaje de petición.

El servidor recibe la petición junto a la cookie y las interpreta.



El navegador guarda las cookies durante un tiempo determinado en el disco duro del cliente.

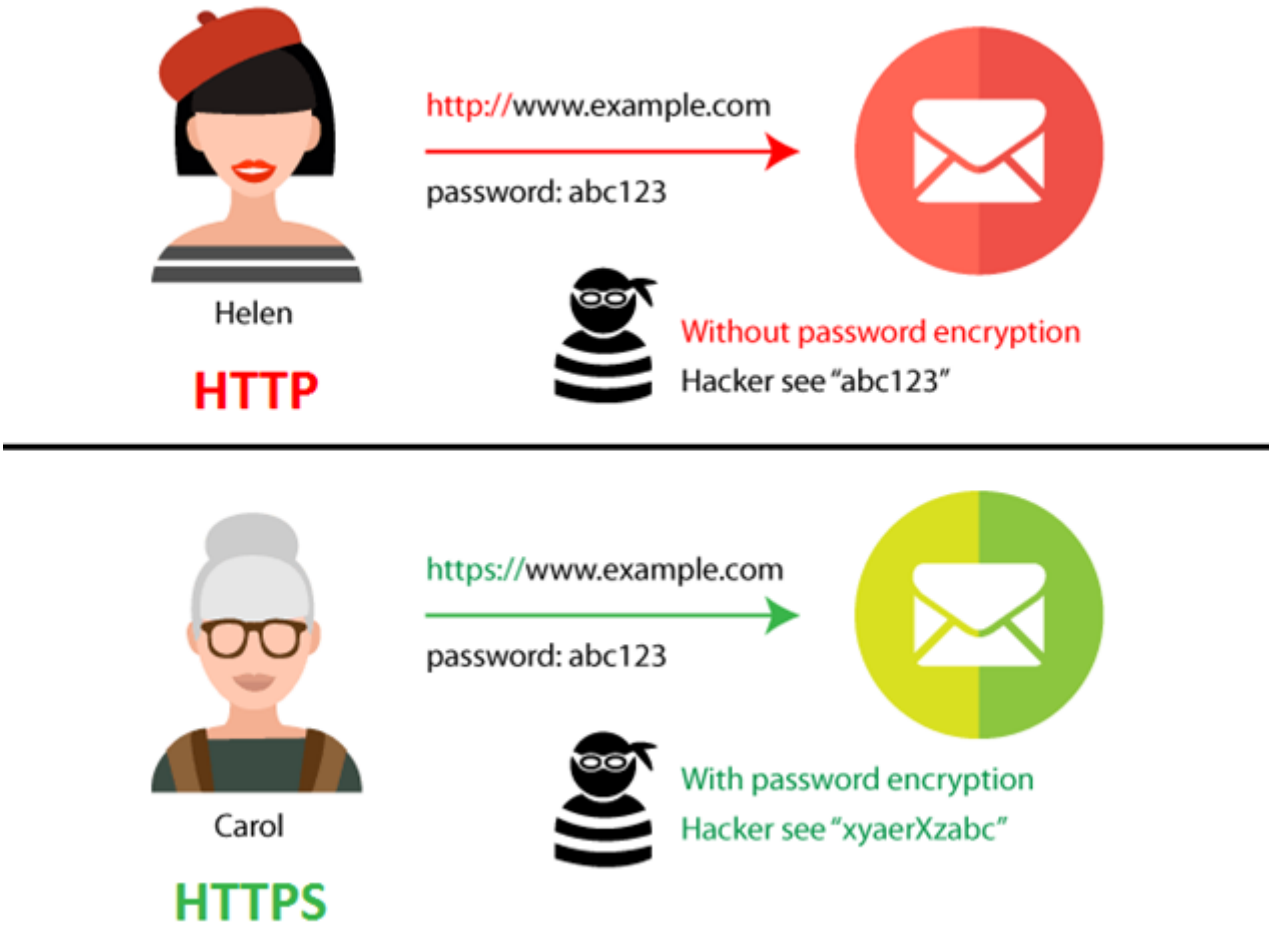
Un campo *expires* especifica cuando *caduca* .

Para eliminar una cookie del disco duro del cliente, el servidor la vuelve a enviar de nuevo, con una fecha de expiración pasada.

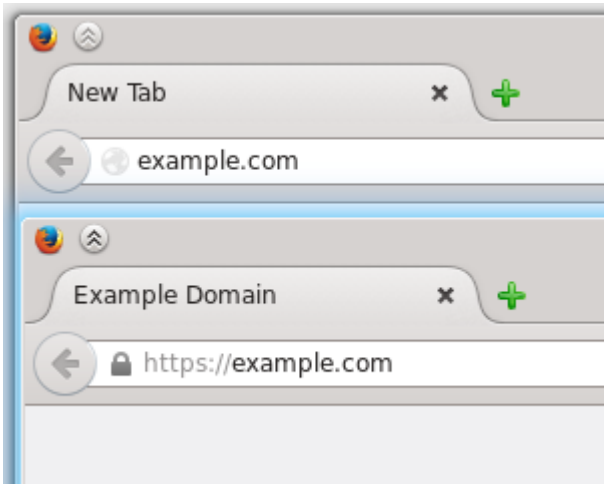
Protocolo HTTPS

El objetivo de este protocolo es realizar una *transferencia* de datos de forma *segura*

Las transacciones bancarias o de pago, o en general cualquier servicio que requiera el envío de datos personales y contraseñas



El formato de la URL es igual al de **HTTP** , excepto que el nombre del protocolo cambia por **HTTPS**



Características

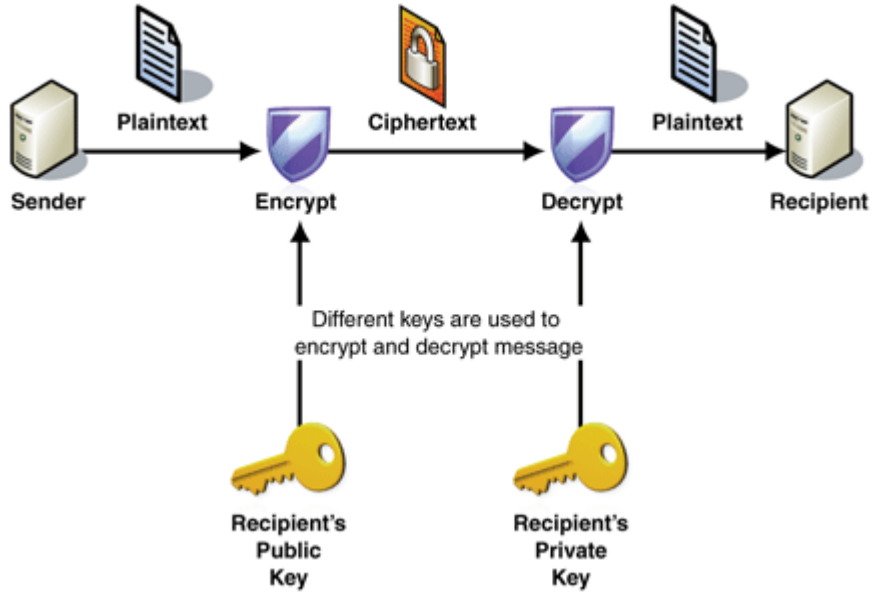
- Necesita de un certificado X.509
- Dos tipos de certificados
  - Autofirmados
  - Firmados por una CA (autoridad de certificación)
- Permite que el tráfico de datos viaje cifrado
- Permite autenticación si el certificado está firmado por una CA
- HTTPS = HTTP + SSL/TLS

- Negociación SSL dependiente de la IP. No se pueden servir diferentes certificados con la misma IP

Puerto por defecto: *443* (80 en HTTP)

HTTPS opera *cifrando* el mensaje antes de ser enviado y decodificándolo antes de que llegue a su destino.

Para cifrar los mensajes se utiliza un _sistema de clave pública _ (pública-privada)



Obtener un certificado

- Make-ssl-cert (paquete ssl-cert)
- Openssl
- Xca

Obtener un certificado

Openssl

```
Openssl
# Generamos clave privada
openssl genrsa -out key 1024

# Generamos CSR y firmamos con nuestra clave privada
openssl req -new -key key -out csr
openssl x509 -req -days 365 -in csr -signkey key -out crt

# Copiamos clave y certificado en el destino
cat key crt > /etc/ssl/private/nombre-sitio.pem
CSR = solicitud de firma de certificados
```

Características

El principal *inconveniente* que presenta es que si alguien consiguiera corromper la clave pública de nuestro servidor web, cambiándolo por la suya, podría descodificar los mensajes del emisor.

HTTPS no nos garantiza que quien hay detrás de la página es quien dice ser, únicamente que la información va cifrada hasta el servidor.

Si nos conectamos por HTTPS a una página fraudulenta, obtendrán nuestros datos igualmente.