

# Protocolo 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

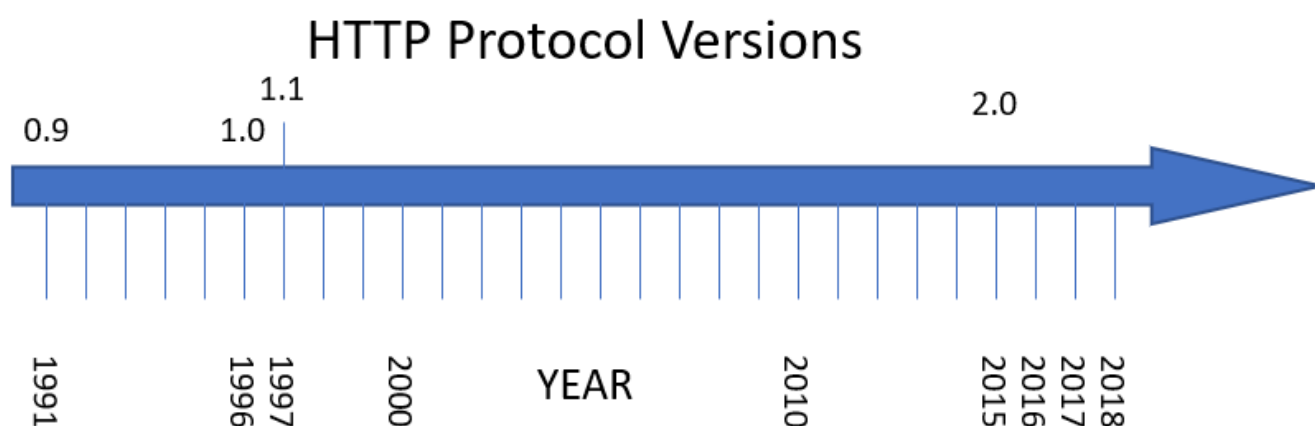
| Name   | Used for                | Example                             |
|--------|-------------------------|-------------------------------------|
| http   | Hypertext (HTML)        | http://www.ee.uwa.edu/~rob/         |
| https  | Hypertext with security | https://www.bank.com/accounts/      |
| 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                       |

## El 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

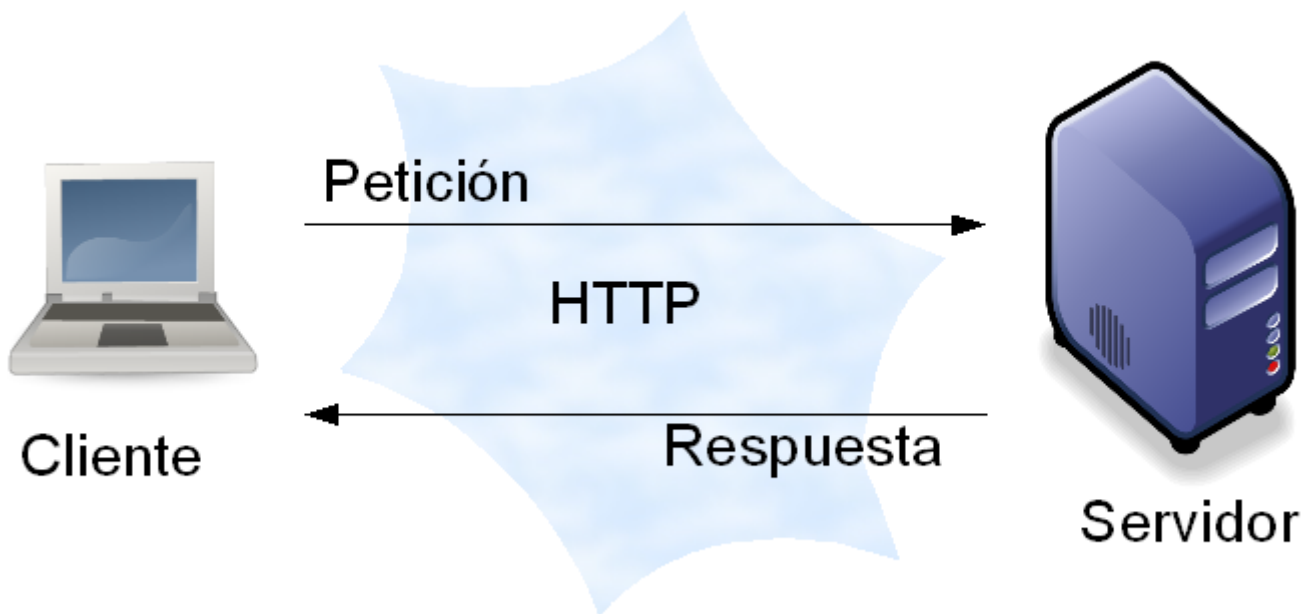
Las *versiones* de HTTP son HTTP/1.0, 1.1, 1.2 y 2



## Cientes y servidores

Los **clientes web** son máquinas que acceden a la información en la web a través de un **navegador**.

El **servidor web**, por otro lado, proporciona contenido web a través del protocolo HTTP. Para ello, se debe instalar algún programa que actúe como servidor web en la máquina, como por ejemplo Apache.



## ¿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.

## El servicio HTTP

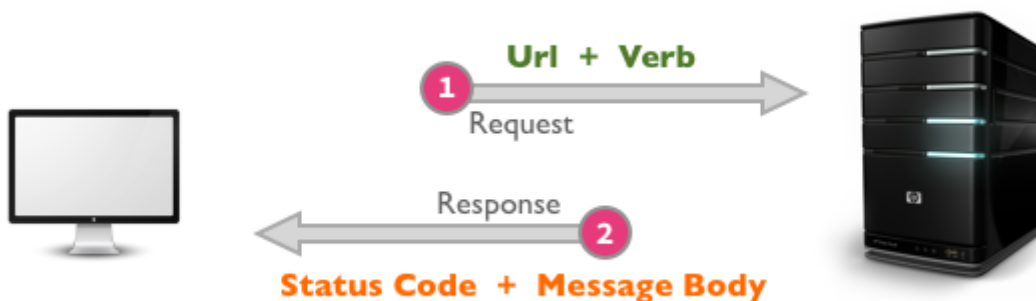
- **Las principales características de HTTP son:**
- Utiliza una estructura \_cliente/servidor. \_ Los servidores alojan las páginas web, y los clientes acceden a ellas a través de un navegador.
- Para visualizar los datos a través de HTTP se requiere un *navegador* \_ \_web. \_ El navegador analiza 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.

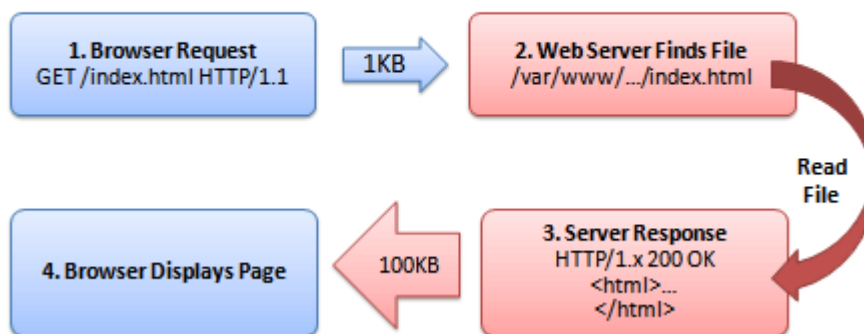
## Ejemplo de sesión

Una **sesión** HTTP consiste en una secuencia de transacciones solicitud y respuesta, en la cual el usuario escribe una dirección en su navegador, el cual realiza una consulta **DNS** para averiguar la dirección IP asociada a la URL y luego intenta establecer una conexión **TCP** al **puerto 80**.

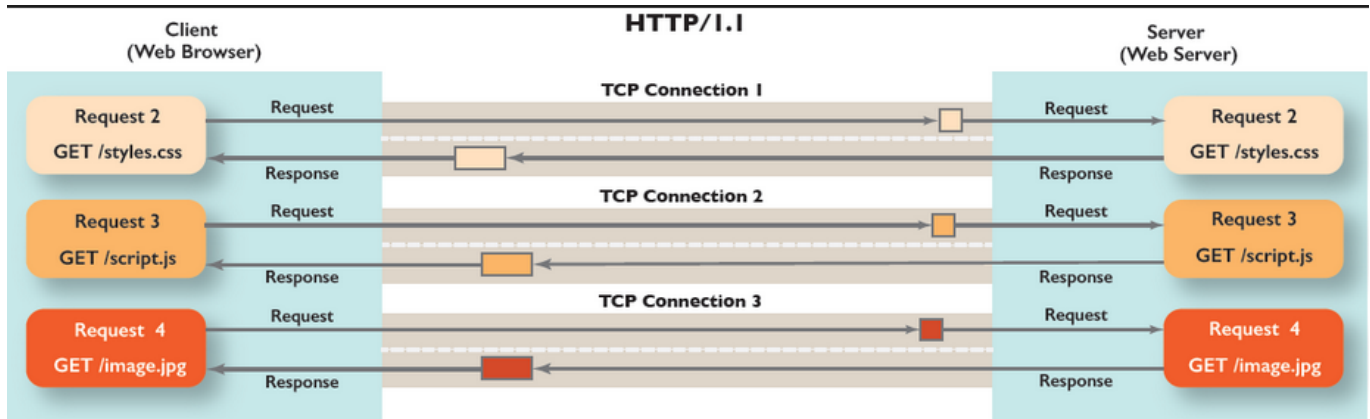
Después de establecer la conexión, el navegador envía la **petición HTTP** solicitando la URL; el servidor responde con un **código de estado** y el **recurso** solicitado. Este proceso es completamente transparente al usuario, que únicamente ve la carga de la página, que en realidad fue realizada por el navegador.



## HTTP Request and Response



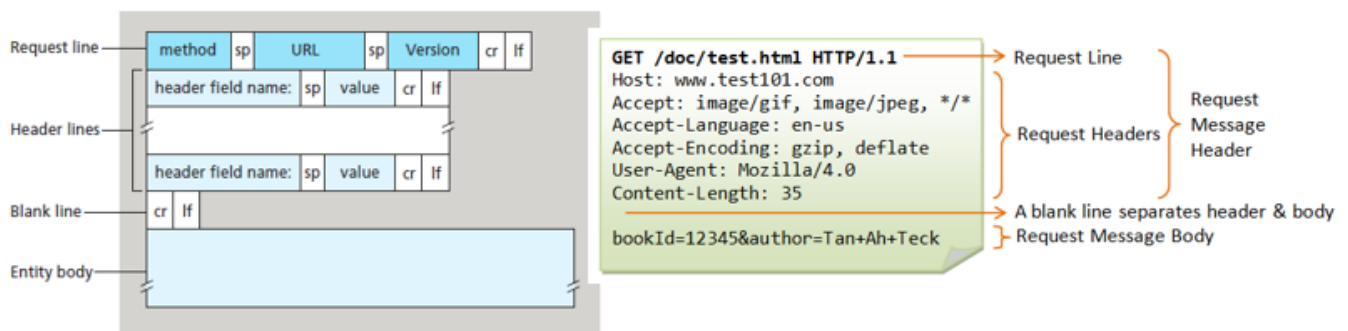
Cuando una página web es visitada, el navegador debe solicitar todos los **objetos externos** que la componen, como una hoja de estilos CSS para definir la apariencia de la página, un archivo de script JavaScript para hacerla dinámica, e incluso una imagen en formato png.



## 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.



## 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)
- *Método 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

The diagram shows an HTTP request format with the following components and annotations:

- request line** (GET, POST, HEAD commands): Points to the first line of the request: `GET /~zasharif/Web/SE432/SE432.html HTTP/1.1\r\n`.
- header lines**: Points to the subsequent lines: `Host: www.just.edu.jo\r\n`, `User-Agent: Firefox/3.6.10\r\n`, `Accept: text/html,application/xhtml+xml\r\n`, `Accept-Language: en-us,en;q=0.5\r\n`, `Accept-Encoding: gzip,deflate\r\n`, `Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n`, `Keep-Alive: 115\r\n`, and `Connection: keep-alive\r\n`.
- carriage return, line feed at start of line indicates end of header lines**: Points to the `\r\n` sequence at the end of the last header line.
- carriage return character** and **line-feed character**: Point to the `\r` and `\n` characters respectively in the `\r\n` sequence.

## Respuestas HTTP

La petición realizada fue satisfactoria ya que el servidor respondió con un código de estado 200 OK, indicando que el recurso fue encontrado. Además, el servidor informó la fecha y hora de respuesta, la versión y módulos del servidor web y el tipo de contenido (text/html).

Finalmente, el servidor entregó el contenido pedido, en este caso el 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 códigos de estado HTTP son números de 3 dígitos que indican el resultado de una solicitud. Los primeros dígitos indican el estado general de la solicitud, mientras que los otros 2 dígitos proporcionan información adicional. Por ejemplo:

- el código de estado 200 OK significa que la solicitud se completó exitosamente y los datos se devuelven, mientras que
- el código de estado 301 Moved Permanently indica que el recurso solicitado se ha movido y se especifica la nueva ubicación.
- El código de estado 302 Moved Temporarily significa que la URL está temporalmente fuera de servicio y se especifica una nueva ubicación.
- Los códigos de estado 400 Bad Request indican un error de sintaxis en la solicitud, mientras que el código de estado 403 Forbidden significa que el usuario no tiene permiso para realizar la solicitud. El código de estado 404 Not Found indica que el documento solicitado no existe, mientras que el código de estado 408 Request Timeout indica que la petición tardó demasiado en completarse por algún motivo.
- El código de estado 505 HTTP Version Not Supported significa que la versión HTTP solicitada no está soportada.

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
```

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., <i>gzip</i> )      |
| 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 suscripción
  - Comercio electrónico
- ¿Cómo podríamos *rastrear* a los usuarios?

¿Podríamos rastrear a los usuarios diferenciándolos por IP?

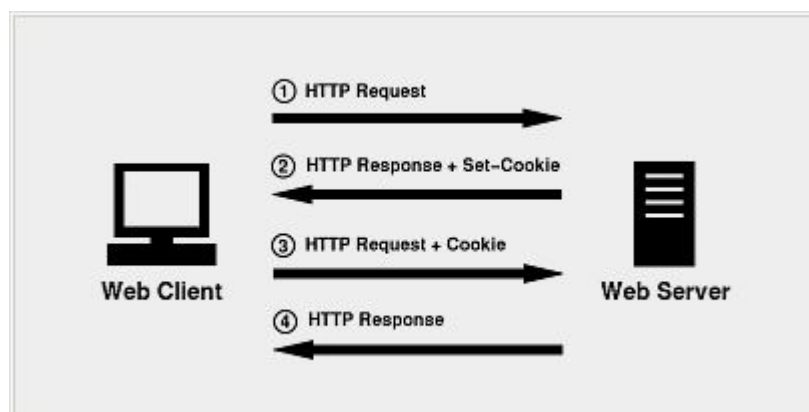
En las casas y oficinas, varios equipos comparten la misma IP pública. Estos equipos se ven desde fuera como uno solo.

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

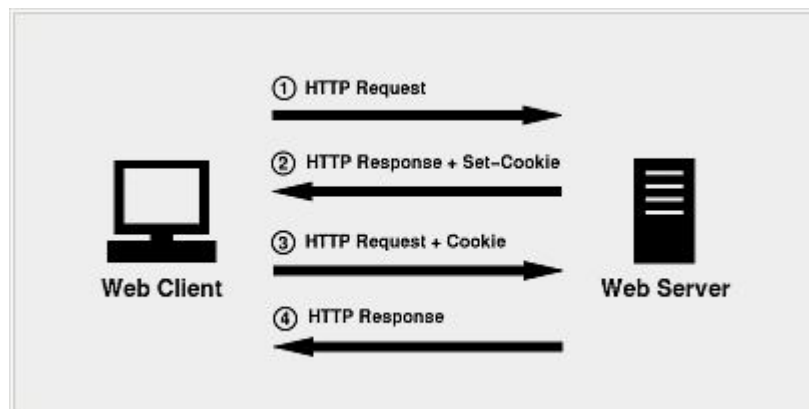
Por ejemplo se puede **guardar** el identificador de sesión, el idioma, el tamaño de letra, etc.

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.

