# El protocolo HTTP

Jesús Arias Fisteus

**Aplicaciones Web (2020/21)**

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

# El protocolo HTTP

**HTTP (Hypertext Transfer Protocol)** es un protocolo sin estado de la capa de aplicación para sistemas de información de hipertexto distribuidos y colaborativos.

# El protocolo HTTP

HTTP se basa en el envío de mensajes sobre el protocolo de transporte TCP:

- El cliente envía un **mensaje de petición** a un servidor, solicitando realizar una acción sobre un **recurso** determinado (habitualmente, obtener el recurso).
- El servidor envía un **mensaje de respuesta** a la petición del cliente (habitualmente, incluyendo el recurso solicitado).

# Versiones del protocolo

Las versiones más utilizadas actualmente son:

- **HTTP/1.1**: versión utilizada mayoritariamente desde finales de los 90.
- **HTTP/2**: versión desplegada en los últimos años. Mejora la eficiencia mediante codificación binaria de mensajes, compresión de cabeceras, multiplexación de múltiples peticiones/respuestas sobre una única conexión TCP, peticiones iniciadas por el servidor, etc.

*(Ambas versiones del protocolo son compatibles en términos de semántica y estructura de los mensajes, cambiando principalmente la codificación de los mensajes y el transporte de los mismos mediante conexiones TCP.)*

# Identificadores de recursos

Los recursos se identifican en HTTP mediante **identificadores uniformes de recurso** (URI, *Uniform Resource Identifier*).

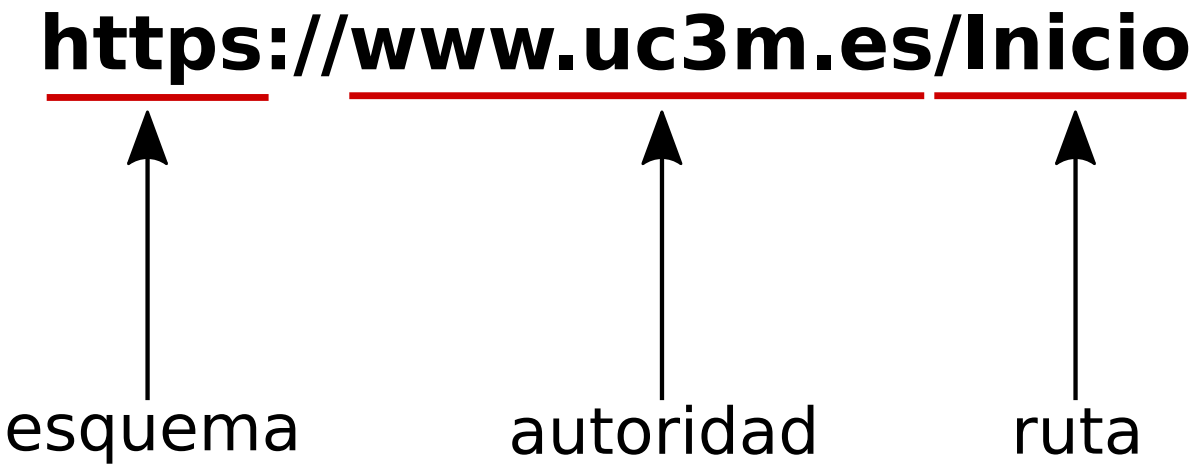
Un **URI** es una secuencia de caracteres compacta que identifica a un recurso abstracto o físico, utilizado en múltiples protocolos y aplicaciones.

Un URI que además proporciona la información necesaria para localizar y acceder al recurso se denomina **localizador uniforme de recurso** (URL, *Uniform Resource Locator*).

# Partes de un URI

- Esquema: hace referencia al nombre de un *esquema*, que define cómo se asignan los identificadores en su ámbito. Los esquemas habituales en la Web serán `http` y `https`.
- Autoridad: elemento de una autoridad jerárquica de asignación de nombres, típicamente basado en un nombre de dominio de DNS o una dirección de red (IP, IPv6) y, opcionalmente, un número de puerto.
- Ruta: elemento que identifica un recurso en el ámbito del esquema y autoridad proporcionados, típicamente organizado jerárquicamente en fragmentos separados por "/".
- Consulta: datos no jerárquicos que permiten, en combinación en la ruta, identificar el recurso. Es habitual representarlo como uno o más pares nombre/valor.
- Identificador de fragmento: identifica un recurso secundario en el contexto del recurso primario como, por ejemplo, un fragmento concreto de una página Web.

# Partes de un URI: ejemplo



# Ejemplos de URI en HTTP con consulta

`https://aulaglobal.uc3m.es/course/view.php?id=91019`

`https://www.google.com/search?q=madrid&tbm=isch`

# Ejemplos de URIs en HTTP con fragmento

```
http://example.com/manual#cap3
```

```
http://example.com/manual?lang=es#cap3
```

# Caracteres reservados

- Un URI puede contener solo letras de la tabla US-ASCII, dígitos y unos pocos símbolos gráficos (“-”, “.”, “_” y “~”), además de los caracteres reservados utilizados entre otros para delimitar elementos (“:”, “/”, “?”, “#”, “[”, “]”, “@”, etc.)
- Otros caracteres, así como los caracteres reservados cuando se utilicen para representar datos y no como delimitadores, deben ser codificados con **codificación de URL**.

# Codificación de URL

- Se codifica cada carácter no permitido en un URI como una secuencia de octetos, y cada octeto se representa mediante el símbolo "%" seguido del valor del octeto codificado con dos caracteres hexadecimales.
- Por ejemplo:
  - "path=docs/index.html" se codifica como "path=docs%2Findex.html" (el carácter "/" se codifica mediante el octeto 2F en ASCII, UTF-8 y la mayoría de sistemas de codificación de caracteres).
  - "q=evaluación" se codifica como "q=evaluaci%C3%B3n" (el carácter "ó" se codifica en UTF-8 mediante la secuencia de octetos C3 y B3).

# Métodos de HTTP

Los mensajes de petición especifican un **método**, que define la acción a realizar sobre le recurso.

# Métodos de HTTP

Los principales métodos utilizados por las aplicaciones Web son:

- **GET**: obtener el recurso.
- **POST**: realizar un procesado (de naturaleza específica para el recurso) basado en los datos que se envían con la petición.

*(Otros métodos definidos por el estándar son: HEAD, PUT, DELETE, CONNECT, OPTIONS y TRACE.)*

# Peticiones con método GET

Las peticiones **GET**:

- Se utilizan para obtener el contenido de recursos (páginas HTML, imágenes, etc.)
- Son generadas por los navegadores Web, entre otros, cuando se introduce un URL en la barra de direcciones, se pincha en un hipervínculo, se deben pedir recursos adicionales ligados a una página HTML recibida, se envían algunos formularios, etc.
- Están sujetas al uso de cachés para optimizar el uso de recursos.
- Se consideran seguras, esto es, no deben tener efectos no deseados en el servidor, estado de la aplicación, etc.

# Peticiones con método POST

Las peticiones **POST**:

- Se utilizan para realizar *acciones* (autenticar a un usuario, añadir un producto al carro de la compra, confirmar un pedido en una tienda, subir un nuevo mensaje a una red social, etc.).
- Son generadas por los navegadores Web cuando se envían algunos formularios.
- No están sujetas al uso de cachés.
- Pueden no ser seguras. Entre otros problemas potenciales, repetir la petición podría tener efectos no deseados (por ejemplo, realizar un mismo pedido dos veces).

# Componentes de una petición

Una petición HTTP incluye:

- **URL del recurso** (sin esquema ni autoridad).
- **Método**.
- **Cabeceras de la petición**: datos adicionales acerca de cómo debe ser procesada la petición.
- **Cuerpo de la petición** (solo con algunos métodos): datos a ser procesados por el servidor.

# Cuerpo de la petición

El cuerpo de la petición:

- No puede ser incluido en peticiones GET.
- Incluye, en un petición POST, los datos a utilizar en el servidor para procesarla.
- Se suele acompañar de las cabeceras `Content-Type` y `Content-Length` de la petición.

# Ejemplo de petición en HTTP/1.1

```
GET /Inicio HTTP/1.1
Host: www.uc3m.es
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Chrome/62.0.3202.89
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9,en;q=0.8,en-US;q=0.7
```

# Componentes de una respuesta

Una respuesta HTTP incluye:

- **Un estado**: código numérico indicando el resultado del procesado de la petición.
- **Cabeceras de la respuesta**: datos adicionales acerca de cómo debe ser procesada la respuesta.
- **Cuerpo de la respuesta**: representación de la respuesta a la petición, típicamente una página HTML, una imagen, etc.

# Ejemplo de respuesta en HTTP/1.1

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=E26E8...; Domain=www.uc3m.es; HttpOnly
Cache-Control: no-store
Last-Modified: Fri, 10 Nov 2017 11:44:28 CET
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Fri, 10 Nov 2017 10:44:28 GMT

<!DOCTYPE html>
<html lang="es" class="no-js">
<head>
  <title>Inicio | UC3M</title>
(...)
```

# Principales cabeceras comunes a peticiones y respuestas

- **`Connection`**: informa al otro extremo de si se debe cerrar o no la conexión TCP una vez completada la respuesta a la petición.
- **`Content-Encoding`**: tipo de codificación (típicamente compresión) aplicado al cuerpo de la petición o respuesta.
- **`Content-Length`**: longitud en *bytes* del cuerpo del mensaje.
- **`Content-Type`**: tipo MIME del cuerpo del mensaje (por ejemplo, `text/html`).

# Principales cabeceras de las peticiones

- **`Accept`**: preferencias del cliente acerca de los tipos de contenido a recibir.
- **`Accept-Encoding`**: preferencias del cliente acerca de los tipos de codificación del cuerpo de la respuesta a recibir (típicamente compresión).
- **`Cookie`**: envío de *cookies* de vuelta al servidor.
- **`Host`**: campo de autoridad del URL que se pide.
- **`If-Modified-Since`**: sello temporal de la última versión del recurso en caché del cliente.
- **`If-None-Match`**: valor **`ETag`** recibido con la última versión del recurso en caché del cliente.
- **`Referer`**: URL desde el cual se originó la petición actual.
- **`User-Agent`**: información (nombre, versión, etc.) acerca del *software* del cliente.

# Principales cabeceras de las respuestas

- **`Cache-Control`**: directivas acerca del almacenamiento de la respuesta en caché.
- **`ETag`**: código que identifica el contenido actual del recurso.
- **`Expires`**: indicación de hasta cuándo se puede guardar el recurso en caché.
- **`Vary`**: listado de cabeceras de la petición cuyos valores pueden afectar a que cambie el contenido de un recurso.
- **`Location`**: en respuestas de redirección, nuevo URL a pedir por el cliente.
- **`Server`**: información (nombre, versión, etc.) acerca del *software* del servidor.
- **`Set-Cookie`**: establece *cookies* que el cliente debe enviar en futuras peticiones.

# Códigos de estado de las respuestas HTTP

Cinco tipos de código de estado:

- **1XX**: de información.
- **2XX**: petición procesada con éxito.
- **3XX**: redirección a otro recurso.
- **4XX**: error en la petición.
- **5XX**: error en el servidor.

# Códigos de estado de las respuestas HTTP

| Código | Razón | Significado |
|---|---|---|
| 200 | OK | Petición procesada con éxito. |
| 301 | Moved Permanently | Recurso movido a otro URL (cabecera `Location`), que el cliente debe usar siempre a partir de ahora. |
| 302 | Found | Recurso movido temporalmente a otro URL (cabecera `Location`). |
| 303 | See Other | Se debe cargar otro recurso (página de confirmación, progreso, etc.) con método `GET`. |
| 304 | Not Modified | El cliente puede usar su versión del recurso en caché. |
| 400 | Bad Request | El cliente envió una petición HTTP inválida (sintaxis, etc.). |
| 403 | Forbidden | El cliente no puede acceder al recurso. |
| 404 | Not Found | No existe un recurso con la ruta dada. |
| 405 | Method Not Allowed | El recurso no admite el método indicado en la petición. |
| 500 | Internal Server Error | Error en el servidor al procesar la petición. |

# *Cookies*

HTTP es un protocolo sin estado, esto es, cada petición es independiente de las demás.

Las **cookies** permiten mantener estado: consisten en pequeñas cantidades de datos asociados a un nombre que el servidor genera y envía al cliente en mensajes de respuesta para que este las incluya en sucesivas peticiones.

# Estructura de las *cookies*

Una *cookie* se representa como una pequeña cadena de texto que contiene:

- Un **nombre**: un servidor puede establecer varias *cookies* con distintos nombres.
- Un **valor**: los datos de la *cookie* en sí mismos.
- Atributos:
  - `Domain` y `Path`: definen en qué peticiones, por autoridad y ruta, el cliente enviará la *cookie* al servidor.
  - `Expires` y `Max-Age`: definen cuándo la *cookie* debe dejar de ser utilizada por el cliente. Si no se especifica ninguno, se elimina al cierre del navegador.
  - `Secure`: la *cookie* solo puede ser enviada por canales seguros (HTTPS típicamente).
  - `HttpOnly`: solo se debe enviar o permitir acceso a la *cookie* a través de HTTP o HTTPS. Por ejemplo, no debe ser accesible a código *JavaScript*.

# Ejemplos de uso de *cookies*

**Creación de cookies** (en respuestas HTTP):

```
Set-Cookie: sid=4RT67aY...;
            Expires=Thu, 13 Feb 2020 21:47:38 GMT;
            Path=/; Domain=.example.com; Secure; HttpOnly
```

**Envío de cookies** (en peticiones HTTP):

```
Cookie: sid=4RT67aY...
```

# Aplicaciones de las *cookies*

- Algunos usos típicos de las *cookies* son los siguientes:
  - **Gestión de sesiones**: el usuario se autentica al principio para crear una sesión (el servidor envía una *cookie* con un *token* de sesión). El servidor identifica peticiones subsiguientes como parte de la misma sesión porque incluyen este mismo *token* de sesión.
  - **Almacenamiento de preferencias**: se pueden almacenar en *cookies* las preferencias del usuario para el sitio Web.
  - **Rastreo de usuarios**: los sitios Web pueden utilizar *cookies* para rastrear el comportamiento de los usuarios (cuando terceros hacen este rastreo, por ejemplo con fines comerciales, se puede considerar que su uso es abusivo).

# HTTP sobre TLS (HTTPS)

**HTTP sobre TLS**, también llamado **HTTPS** (*Hypertext Transfer Protocol Secure*), define cómo se transporta HTTP sobre un canal seguro TLS (*Transport Layer Security*).

# Propiedades de seguridad de HTTPS

El uso de HTTP sobre TLS proporciona las siguientes propiedades de seguridad:

- **Autenticación**: se autentica siempre al servidor y, opcionalmente, al cliente.
- **Confidencialidad**: los datos enviados por el canal seguro una vez este se ha establecido son solo visibles por los dos extremos del mismo.
- **Integridad**: cualquier modificación de los datos enviados por el canal seguro una vez se ha establecido este será detectada.

# HTTP/2

**HTTP/2** fue definido para corregir algunos aspectos de HTTP/1.1 que impactan negativamente en el rendimiento las aplicaciones *Web* actuales. En concreto, optimiza cómo se transportan los mensajes de HTTP sobre la conexión subyacente.

HTTP/2 mantiene la semántica de los mensajes de HTTP/1.1 (estructura, cabeceras, etc.).

# Principales cambios en HTTP/2

- La unidad básica del protocolo es la **trama**, que **se codifica en formato binario** para agilizar su procesamiento. Cada mensaje se encapsula en una o varias tramas.
- Distintas peticiones y sus respuestas se **multiplexan como flujos independientes en una única conexión**, sin que el retraso en el procesamiento de ciertas peticiones afecte al resto de peticiones concurrentes.
- Se añaden al protocolo mecanismos de **control de flujo** y de **establecimiento de prioridades** entre los distintos flujos.
- Un servidor puede enviar respuestas al cliente sin que este haya enviado previamente las peticiones correspondientes (**server push**), anticipando probables peticiones futuras del cliente.
- Se aplica un algoritmo de **compresión de las cabeceras** de los mensajes (HPACK) para reducir su tamaño, dado el alto grado de redundancia que estas presentan.

# Referencias


- Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. IETF RFC 7230. Junio de 2014.
- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. IETF RFC 7231. Junio de 2014.
- Uniform Resource Identifier (URI): Generic Syntax. IETF RFC 3986. Enero de 2005.
- HTTP State Management Mechanism. IETF RFC 6265. Abril de 2011.
- Hypertext Transfer Protocol Version 2 (HTTP/2). IETF RFC 7540. Mayo de 2015.

# Otros recursos

- MDN Web Docs, "Web Technology for Developers: HTTP"