Que es HTTP?

Hypertext Transfer Protocol (**HTTP**) (o Protocolo de Transferencia de Hipertexto en español) es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también.

Pero ¿qué significa ***hipertexto***, ese término que forma parte de las siglas HTTP y HTML? Se trata de un concepto que en realidad todos conocemos: el enlace a otros archivos, como los [hiperenlaces](https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/hipervinculos-que-son-realmente-y-para-que-se-utilizan/) que se usan en las páginas web para redirigir a otras páginas.

| **Protocolo de transferencia de hipertexto** | | |
| --- | --- | --- |
| HTTP logo.svg | | |
| **Familia** | [Familia de protocolos de Internet](https://es.wikipedia.org/wiki/Familia_de_protocolos_de_Internet) | |
| **Función** | Transferencia de [hipertexto](https://es.wikipedia.org/wiki/Hipertexto) | |
| **Última versión** | 1.3 (2018) | |
| **Puertos** | 80/TCP | |
| **Ubicación en la pila de protocolos** | | |
|  | | |

Aplicacion: HTTP

Transporte:TCP

Red:IP

Que funcion cumple HTTP:

Cuando escribes una dirección web en tu navegador y se abre la página que deseas, es porque tu navegador se ha comunicado con el servidor web por HTTP. Dicho de otra manera, el protocolo HTTP es el código o lenguaje en el que el navegador le comunica al servidor qué página quiere visualizar.

Como funciona HTTP:

La manera más fácil de explicar cómo funciona HTTP es describiendo cómo se abre una página web:

1. En la barra de direcciones del navegador, el usuario teclea [*example.com*](http://example.com/).
2. El navegador envía esa solicitud, es decir, la [**petición HTTP**](https://www.ionos.mx/digitalguide/hosting/cuestiones-tecnicas/http-request/), al servidor web que administre el dominio *example.com*. Normalmente, la solicitud del cliente dice algo así como “Envíame este archivo”, pero también puede ser simplemente “¿Tienes este archivo?”.
3. El servidor web recibe la solicitud HTTP, busca el archivo en cuestión (en nuestro ejemplo, la página de inicio de *example.com*, que corresponde al archivo *index.html*) y envía en primer lugar una cabecera o ***header***. Esta cabecera le comunica al cliente, mediante un código de estado, el resultado de la búsqueda. Para conocer más [detalles acerca de los códigos de estado](https://www.ionos.mx/digitalguide/hosting/cuestiones-tecnicas/una-mirada-a-los-codigos-de-estado-http-mas-comunes/), no te pierdas nuestro artículo al respecto.
4. Si se ha encontrado el archivo solicitado y el cliente ha solicitado recibirlo (y no solo saber si existe), el servidor envía, tras el *header*, el ***message body o*** cuerpo del mensaje, es decir, el contenido solicitado: en nuestro ejemplo, el archivo *index.html*.
5. El navegador recibe el archivo y lo abre en forma de página web.

1)Encabezado HTTP:

Los **HTTP headers** son la parte central de los HTTP requests y responses, y transmiten información acerca del navegador del cliente, de la página solicitada, del servidor, etc.

Cada vez que visitas cualquier sitio, puedes ver los **headers del request enviado**:

GET php.net HTTP/1.1 \*\*Accept\*\*: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

La primera línea es la **línea del request**, que contiene su información básica (método HTTP, URL y versión). Lo demás son headers HTTP.

Una vez se envía ese request al servidor, éste responde con un **HTTP response**:

HTTP/1.1 200 OK

La primera línea es la **barra de estado** (versión HTTP y código de respuesta), seguido de los HTTP headers, hasta una línea vacía. Bajo la línea vacía se encuentra el contenido (en este caso contenido html).

Cuando se observa el código fuente de un sitio web los headers no aparecen, pero se han recibido.

Los HTTP requests se envían y reciben para muchas otras cosas, imágenes, archivos CSS, archivos JavaScript, etc. Cuando cargas una página web se envían y reciben por cada *resource*.

Para ver los headers se puede hacer mediante Firebug en **Firefox**, las Developer Tools de **Chrome**, **curl**, o con funciones de PHP ([*getallheaders()*](http://php.net/manual/en/function.getallheaders.php) o **$\_SERVER** para obtener los **request headers** y \_[headers\_list()](http://www.php.net/manual/en/function.headers-list.php)\_ para obtener los **response headers** enviados o por enviar).

### **2. Estructura de un HTTP request**

Un HTTP request se compone de:

* **Método**: GET, POST, PUT, etc. Indica que tipo de request es.
* **Path**: la URL que se solicita, donde se encuentra el resource.
* **Protocolo**: contiene HTTP y su versión, actualmente 1.1.

En el ejemplo anterior: *GET php.net HTTP/1.1*

* **Headers**. Son esquemas de ***key***: ***value*** que contienen **información sobre el HTTP request y el navegador**. Aquí también se encuentran los datos de las [cookies](http://diego.com.es/cookies-en-php). La mayoría de los headers son opcionales.
* **Body**. Si se envía información al servidor a través de POST o PUT, ésta va en el body

### **3)Métodos HTTP**

Los métodos más importates de HTTP (especialmente para hacer [aplicaciones REST](http://diego.com.es/introduccion-a-rest-en-php)) son **POST**, **GET**, **PUT**, **DELETE** y **HEAD**.

* **GET**. El método GET se emplea para leer una representación de un ***resource***. En caso de respuesta positiva (200 OK), GET devuelve la representación en un formato concreto: HTML, XML, JSON o imágenes, JavaScript, CSS, etc. En caso de respuesta negativa devuelve 404 (*not found*) o 400 (*bad request*). Por ejemplo en la carga de una página web, primero se carga la ***url*** solicitada:

GET php.net/docs HTTP/1.1

En este caso devolverá **HTML**. Y después los demás ***resources*** como CSS, JS, o imágenes:

GET php.net/images/logo.png HTTP/1.1

Los formularios también pueden usarse con el [método GET](http://diego.com.es/get-y-post-en-php), donde se añaden los ***keys*** y ***values*** buscados a la URL del header:

* **POST**. Aunque se puedan enviar datos a través del método GET, en muchos casos se utiliza POST por las **limitaciones de GET**. En caso de respuesta positiva devuelve 201 (*created*). Los POST requests se envían normalmente con formularios:

El contenido va en el ***body*** del ***request***, no aparece nada en la URL, aunque se envía en el mismo formato que con el método GET. Si se quiere enviar texto largo o cualquier tipo de archivo este es el método apropiado.

Le siguen los headers, donde se incluyen algunas líneas específicas con información de los datos enviados

A los headers le siguen una **línea en blanco** y a continuación el **contenido del *request***:

* **PUT**. Utilizado normalmente para **actualizar contenidos**, pero también pueden **crearlos**. Tampoco muestra ninguna información en la URL. En caso de éxito devuelve 201 (*created*, en caso de que la acción haya creado un elemento) o 204 (*no response*, si el servidor no devuelve ningún contenido). A diferencia de POST es **idempotente**, si se crea o edita un resource con PUT y se hace el mismo request otra vez, el resource todavía está ahí y mantiene el mismo estado que en la primera llamada. Si con una llamada PUT se cambia aunque sea sólo un contador en el resource, la llamada ya no es idempotente, ya que se cambian contenidos.
* **DELETE**. Simplemente elimina un ***resource*** identificado en la **URI**. Si se elimina correctamente devuelve 200 junto con un *body response*, o 204 sin *body*. DELETE, al igual que PUT y GET, también es **idempotente**.
* **HEAD**. Es idéntido a GET, pero el servidor no devuelve el contenido en el **HTTP response**. Cuando se envía un **HEAD request**, significa que sólo se está interesado en el código de respuesta y los **headers HTTP**, no en el propio documento. Con este método el navegador puede comprobar si un documento se ha modificado, por razones de caching. Puede comprobar también directamente si el archivo existe.  
  Por ejemplo, si tienes muchos enlaces en tu sitio web, puedes enviar un **HEAD request** a todos los enlaces para comprobar los que estén rotos. Es bastante más rápido que hacerlo con GET.

### **4. Estructura de un HTTP response**

Una vez que el navegador envía el **HTTP request**, el servidor responde con un **HTTP response**, compuesto por:

* **Protocolo**. Contiene HTTP y su versión, actualmente 1.1.
* **Status code**. El **código de respuesta**, por ejemplo: **200 OK**, que significa que el GET request ha sido satisfactorio y el servidor devolverá los contenidos del documento solicitado. Otro ejemplo es **404 Not Found**, el servidor no ha encontrado el resource solicitado.
* **Headers**. Contienen información sobre el **software del servidor**, cuando se modificó por última vez el *resource* solicitado, el **mime type**, etc. De nuevo la mayoría son opcionales.
* **Body**. Si el servidor devuelve información que no sean *headers* ésta va en el ***body***.3

### **5. Headers HTTP en HTTP requests**

Los headers pueden obtenerse con el array **$\_SERVER** en PHP. La función *getallheaders()* devuelve todos los headers de vez.

os headers más comunes en los HTTP requests son los siguientes:

* Host
* Host: php.net

Es el nombre del host, incluyendo dominio y subdominio si existe. En PHP se obtiene con **$\_SERVER['HTTP\_HOST']** o **$\_SERVER['SERVER\_NAME']**.

* User-Agent
* User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36

El User-Agent contiene información como el **nombre y versión del navegador** y del **sistema operativo** y el **idioma por defecto**. De esta forma los sitios web pueden saber información acerca de los sistemas de los visitantes. Pueden detectar si el usuario está visitando desde un móvil y redireccionarlo a una versión móvil más adecuada para bajas resoluciones. En PHP se encuentra bajo **$\_SERVER['HTTP\_USER\_AGENT']**.

* Accept-Language
* Accept-Language: en-us, en;q=0.5

Este header muestra el lenguaje por defecto del usuario. Si el sitio web tiene diferentes versiones por idiomas, puede redireccionar al usuario. En PHP: **$\_SERVER['HTTP\_ACCEPT\_LANGUAGE']**.

if(substr($\_SERVER['HTTP\_ACCEPT\_LANGUAGE'], 0, 2) == 'es'){

header('Location: http://es.ejemplo.com');

}

* Accept-Encoding
* Accept-Encoding: gzip, deflate, sdch

Formatos de codificación que soporta el navegador. El servidor web puede enviar el **HTML** resultante en un **formato comprimido**, lo que ahorra hasta un 80% de bandwidth y tiempo de carga. En PHP: **$\_SERVER['HTTP\_ACCEPT\_ENCODING']**.

* If-Modified-Since
* If-Modified-Since: Mon, 9 Nov 2015 08:32:10 GMT

Si no se ha modificado el resource desde esa fecha, el servidor devolverá un código de respuesta 304 Not Modified, sin contenido, ya que el navegador cargará el contenido de la caché. En PHP: **$\_SERVER['HTTP\_IF\_MODIFIED\_SINCE']**.

// Comprobar si el explorador envía el header If-Modified-Since

if(isset($\_SERVER['HTTP\_IF\_MODIFIED\_SINCE'])){

// Comprobar si el cache del navegador coincide con el modify time

if($last\_modify\_time == strtotime($\_SERVER['HTTP\_IF\_MODIFIED\_SINCE'])){

// Enviar un header 304, sin contenido

header("HTTP/1.1 304 Not Modified");

exit;

}

}

También está el header Etag, que se asegura de que la caché es actual.

* Cookie
* Cookie: PHPSESSID=rqwe8f1ew8f1341fiu; usuario=homer

Envía las cookies guardadas en el navegador para ese dominio. En PHP: array **$\_COOKIE**.

* Referer

**Referer**. Contiene la url de referencia. Si un usuario hace click en un enlace, en la página de destino aparecerá como referer la anterior. En PHP: **$\_SERVER['HTTP\_REFERER']**.

if(isset($\_SERVER['HTTP\_REFERER'])){

$url = parse\_url($\_SERVER['HTTP\_REFERER']);

// Comprobar si el visitante viene de google

if($url['host'] == 'www.google.com'){

parse\_str($url['query'], $vars);

echo "Has buscado estas palabras: " . $vars['q'];

}

}

* Authorization

Cuando un sitio web solicita autorización, el navegador abre una ventana de login. Cuando insertas los datos de entrada, el navegador envía otro request, pero esta vez contiene:

Authorization: Basic bXl1c2VyOm15cGFzcw==

El dato incluído está codificado en **base 64**. La función base64\_decode('bXl1aefi128djGFzcw==') devuelve 'myuser:mypass'.

En PHP: **$\_SERVER['PHP\_AUTH\_USER']** y **$\_SERVER['PHP\_AUTH\_PW']**.

### **6. Headers HTTP en HTTP responses**

En PHP se pueden establecer los códigos de respuesta con la función [*header()*](http://php.net/manual/en/function.header.php). PHP ya envía algunos headers automáticamente como para cargar el contenido o establecer cookies. Se pueden ver los headers enviados, o que serán enviados, con la función \_[headers\_list()](http://www.php.net/manual/en/function.headers-list.php)*. Se puede saber si los headers ya se han enviado con* [headers\_sent()](http://www.php.net/manual/en/function.headers-sent.php)\_.

* Cache-Control

Campo usado para especificar las directivas que se deben cumplir por los mecanismos de caché durante la cadena request/response. Estos mecanismos de caché incluyen gateways y proxies que el ISP puede estar utilizando.

Cache-Control: max-age=3600, public

"public" significa que la respuesta puede ser cacheada por cualquiera. "max-age" indica por cuantos segundos es válida. Permitir que el sitio web sea cacheado reduce el consumo de memoria en el servidor y reduce los tiempos de carga.

Se puede evitar el cacheo con "no-cache":

Cache-Control: no-cache

* Content-Type

Indica el **mime-type** del documento. El navegador decide entonces como interpretar los contenidos. Una página html (o PHP con output html):

Content-Type: text/html; charset=UTF-8

"text" es el tipo y "html" es el subtipo del documento. Para una imagen tipo gif:

Content-Type: image/gif

El navegador puede decidir si usar una aplicación externa o una extensión de navegador basándose en el mime-type. Por ejemplo lo siguiente puede cargarse con Adobe Reader:

Content-Type: application/pdf

Cuando se carga directamente, **Apache** normalmente puede detectar el **mime-type** de un documento y enviar el header apropiado. Los navegadores tienen detección automática de mime-types, por si los headers están mal o no están presentes.

En **PHP** se puede utilizar la función \_[finfo\_file()](http://www.php.net/manual/en/function.finfo-file.php)\_ para detectar el mime-type de un archivo.

* Content-Disposition

Este header indica al navegador que abra una caja de descarga de archivos, en lugar de analizar el contenido.

Content-Disposition: attachment; filename="descargar.zip"

Este header anterior debe ir acompañado de este otro:

Content-Type: application/zip

* Content-Length

Cuando el contenido se va a enviar al navegador, el servidor puede indicar el tamaño en bytes:

Content-Length: 123245

Esto es especialmente útil para la descarga de archivos, así el navegador puede calcular el progreso de la descarga.

* Etag

Es otro header que se usa para caching.

Etag: "pub1212441;gz"

El servidor web puede enviar este header con cada documento que envía. El valor puede estar basado en la **última fecha de modificación**, el **tamaño del archivo**, etc. El navegador guarda entonces este valor y cachea el documento. La proxima vez que el navegador solicita el mismo archivo, envía esto en el **HTTP request**:

If-None-Match: "pub1212441;gz"

Si el valor **eTag** del documento concuerda, el servidor enviará un código 304 en lugar de 200, sin contenido. El navegador cargará los contenidos de su caché.

* Last-Modified

Indica la última fecha de modificación del documento en formato GMT.

Last-Modified: Mon, 09 Nov 2015 11:50:11 GMT

En PHP podemos establecerlo así:

$modificarTiempo = filemtime($archivo);

header("Last-Modified: ".gmdate("D, d M Y H:i:s", $modificarTiempo));

* Location

Header utilizado para las redirecciones. Si el código de respuesta es 301 o 302, el servidor debe también enviar este header.

En PHP se puede redirigir a un usuario así:

header('Location: http://www.google.com');

Esta será una **redirección 302**. Si queremos que sea **301**:

header('Location: http://www.google.com', 301);

* Set-Cookie

Cuando un sitio web quiere establecer o actualizar una cookie en tu navegador, utilizará este header.

Set-Cookie: skin=noskin; path=/; domain=.amazon.com; expires=Sun, 21-Nov-2015 14:22:22 GMT

Set-Cookie: session-id=820-1736418-8162394; path=/; domain=.amazon.

Cada cookie se envía en un header separado. En PHP las cookies se establecen con la función *setcookie()*.

Si no se especifica una fecha de expiración, la cookie se eliminará cuando se cierre la ventana del navegador.

* WWW-Authenticate

Un sitio web puede enviar este header para **identificar al usuario a través de HTTP**. Cuando el navegador ve este header, abrirá una ventana de login.

\*\*WWW-Authenticate\*\*: Basic realm="Restricted Area"

En PHP se puede utilizar el siguiente script:

if(!isset($\_SERVER['PHP\_AUTH\_USER'])){

header('WWW-Authenticate: Basic realm="My Realm"');

header('HTTP/1.0 401 Unauthorized');

echo 'Texto a enviar si el usuario cancela';

exit;

} else {

echo "<p>Hola {$\_SERVER['PHP\_AUTH\_USER']}.</p>";

echo "<p>Has introducido {$\_SERVER['PHP\_AUTH\_PW']} como tu contraseña</p>";

}

* Content-Encoding

Header enviado cuando el contenido está comprimido.

Content-Encoding: gzip