

---

# SISTEMAS WEB

## CURSO 2023/2024

### HTTP - HyperText Transfer Protocol

Respuesta: Content-Length vs Transfer-Encoding

Respuesta: Codificación del contenido

Compresión



Web Sistemak by [Oskar Casquero](#) & [María Luz Álvarez](#) is licensed under a [Creative Commons Reconocimiento 4.0 Internacional License](#).

# FUNCIONAMIENTO DE HTTP:

## CONTENT LENGTH Y TRANSFER ENCODING

---

- Un cliente web tiene que leer una respuesta enviada desde un servidor web, necesita saber dónde acaba el cuerpo del mensaje.
  - Una forma que tiene el protocolo HTTP de expresar la longitud del cuerpo del mensaje es con la cabecera "Content-Length".
    - NOTA: la cabecera "Content-Length" requiere conocer el tamaño del contenido que se va a incluir en el cuerpo del mensaje de la respuesta antes de enviarlo.
- A veces hay que enviar una gran cantidad de datos en la respuesta, pero no es posible conocer el tamaño de esos datos hasta que la petición ha sido procesada en su totalidad. Ejemplo: supongamos que los resultados de una solicitud a una base de datos se quieren visualizar en una página web.
  - Si se utiliza la cabecera "Content-Length", hay que conocer el tamaño de los datos que van en el cuerpo del mensaje antes de enviar la respuesta HTTP. Esto, por un lado, supone esperar a finalizar el procesamiento de la solicitud de la base de datos. Por otro lado, para guardar los datos se necesita un gran buffer.
  - El retraso generado por el tiempo necesario para procesar la respuesta antes de su envío influye negativamente en la experiencia del usuario.
- El protocolo HTTP proporciona un mecanismo de ir enviando partes del contenido solicitado por el cliente mientras el servidor web procesa la solicitud: cabecera "**Transfer-Encoding**" con valor "**chunked**".

# FUNCIONAMIENTO DE HTTP:

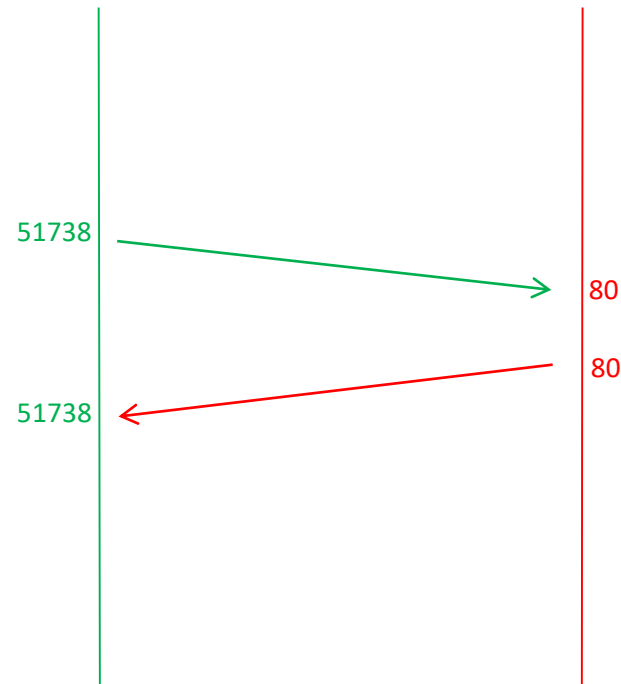
## CONTENT LENGTH Y TRANSFER ENCODING

- Ejemplo de “**Content-Length**”: supongamos que el servidor tiene que enviar el texto “Hello World!!”

### Respuesta HTTP con “Content-Length”

```
HTTP/1.1 200 OK
Date: Thu, 20 Nov 2015 20:25:52 GMT
Last-Modified: Tue, 17 Sep 2015 13:00:02 GMT
ETag: "1a968-3ec-4e693e61bb8b6"
Content-Length: 13
Content-Type: text/plain

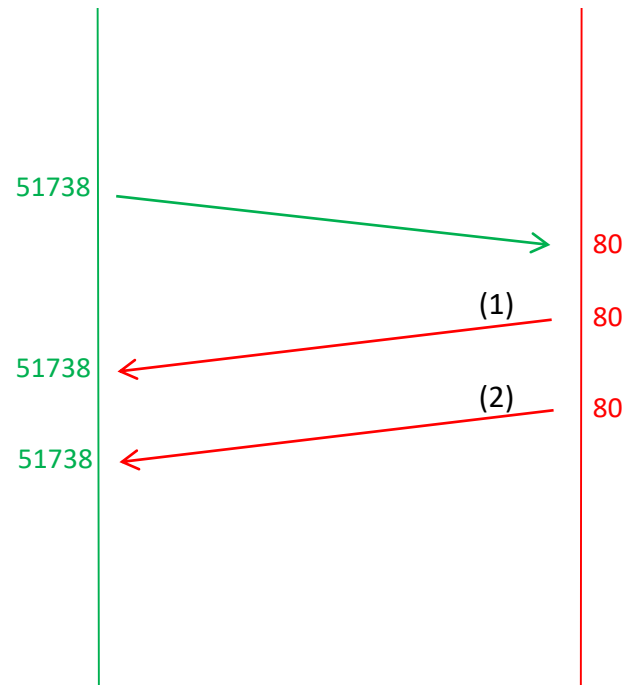
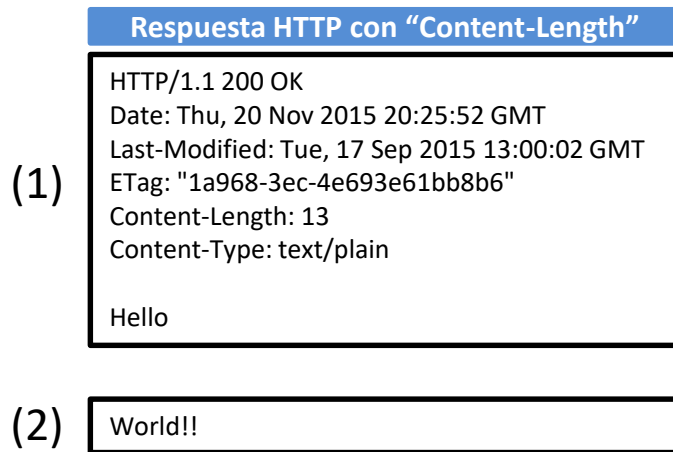
Hello World!!
```



# FUNCIONAMIENTO DE HTTP:

## CONTENT LENGTH Y TRANSFER ENCODING

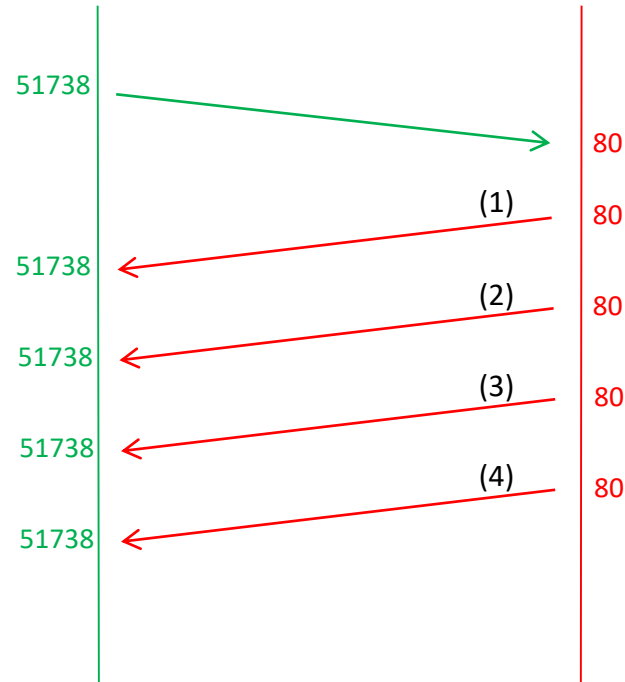
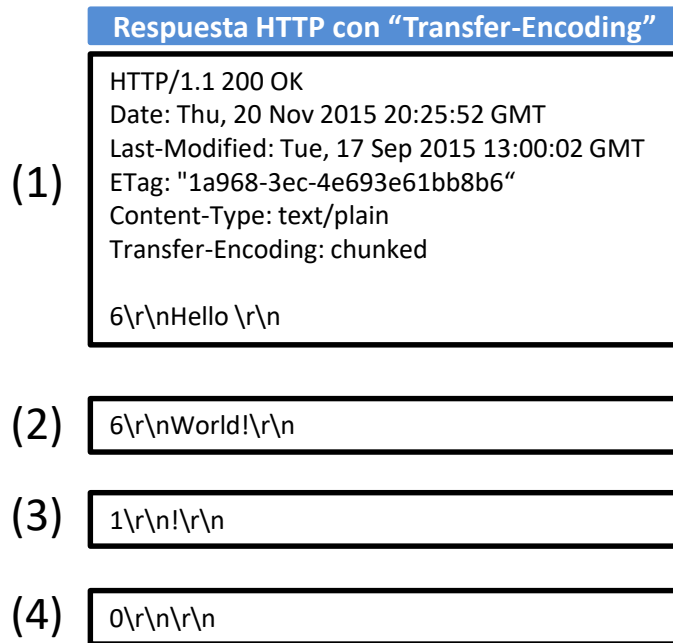
- NOTA: si el contenido de una respuesta es mayor que la MTU (Maximum Transmission Unit) de la capa TCP, el mensaje HTTP se envía en varios segmentos.
- Ejemplo: supongamos que la MTU (Maximum Transmission Unit) de la capa TCP son 187 octetos.



# FUNCIONAMIENTO DE HTTP:

## CONTENT LENGTH Y TRANSFER ENCODING

- Ejemplo “**Transfer-Encoding: chunked**”: supongamos que el servidor tiene que enviar el texto “Hello World!!”



La longitud de cada trozo se indica en hexadecimal.

Los trozos 2, 3 y 4 no llevan cabeceras HTTP. Van directamente encapsulados en segmentos TCP.

# FUNCIONAMIENTO DE HTTP:

## CONTENT LENGTH Y TRANSFER ENCODING

---

- Diferencias entre ambos mecanismos:
  - con "**Content-Length**" los mensajes no se empiezan a enviar hasta que se ha procesado toda la respuesta (latencia alta), mientras que con "**Transfer-Encoding**" los mensajes se envían según se van procesando los trozos (latencia baja)
  - con "**Content-Length**" el troceado de los mensajes (dependiente de la MTU) lo hace la capa TCP, mientras que con "**Transfer-Encoding**" lo hace la propia capa HTTP.

# EJEMPLO:

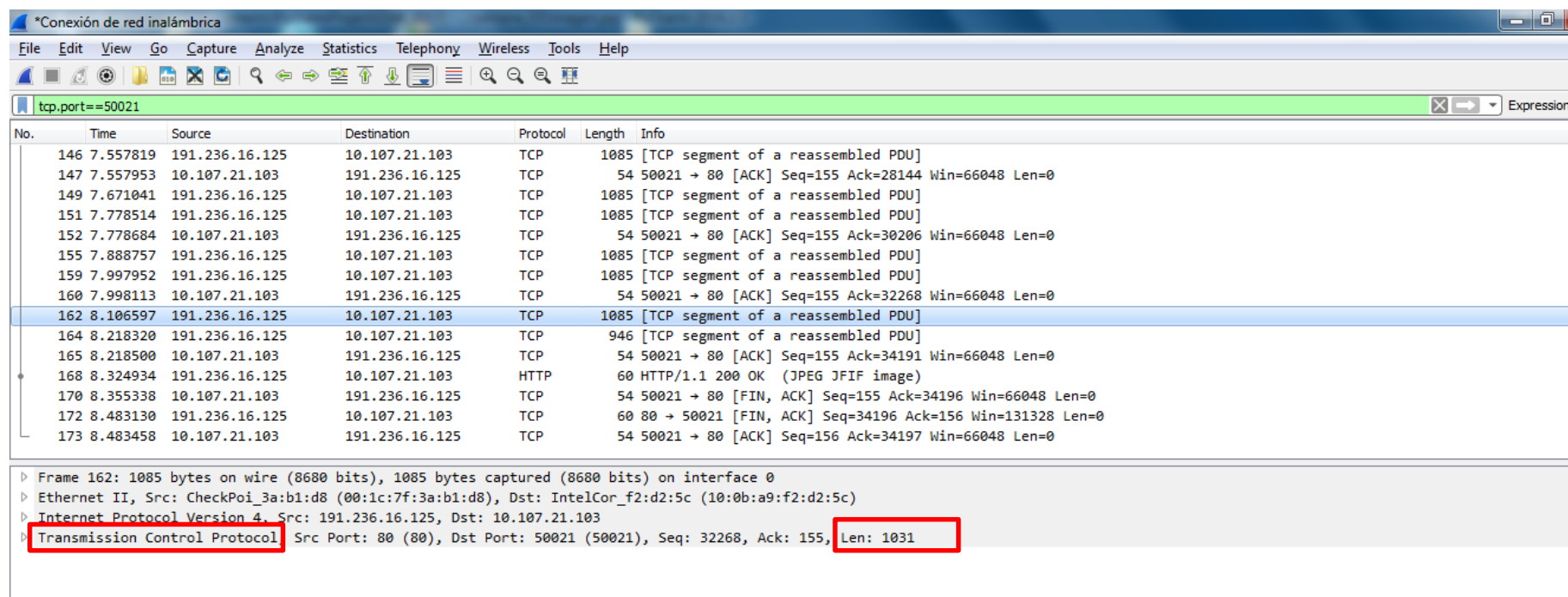
## ANÁLISIS DE LA DESCARGA POR “TROZOS” DE UNA IMAGEN

---

- Haciendo uso del navegador descarga la siguiente imagen:  
<http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx>
- Utilizando Wireshark analiza la captura
  - ¿Cuál es la relación entre el tamaño del campo de datos del segmento TCP y la parte del mensaje HTTP?
    - $3 + 2 + 1024 + 2 = 1031 \rightarrow$  Explicar de dónde salen estos números

# EJEMPLO:

## ANÁLISIS DE LA DESCARGA POR “TROZOS” DE UNA IMAGEN



Frame 162: 1085 bytes on wire (8680 bits), 1085 bytes captured (8680 bits) on interface 0

Ethernet II, Src: CheckPoi\_3a:b1:d8 (00:1c:7f:3a:b1:d8), Dst: IntelCor\_f2:d2:5c (10:0b:a9:f2:d2:5c)

Internet Protocol Version 4, Src: 191.236.16.125, Dst: 10.107.21.103

Transmission Control Protocol, Src Port: 80 (80), Dst Port: 50021 (50021), Seq: 32268, Ack: 155, Len: 1031

0000 10 0b a9 f2 d2 5c 04 .....r ...Z...3  
0010 04 2f 51 11 40 00 6a D.eFF.`G ..^LS...  
0020 15 67 00 50 c3 65 7f .....r6. .2v...].  
0030 02 01 cf fa 00 00 3d ~\*....] .Z....].  
0040 dd 89 a8 87 21 c5 ee 72 e1 ae c6 7a d2 5f 2c 33 .....dR ..pzWD..  
0050 44 dc 65 46 46 f0 60 47 eb cd 5e 4c 53 81 a9 02 ....|.49 x.....v.  
0060 0f 9b 95 19 89 72 36 a5 95 32 76 04 b5 81 5d 81 ...H... Z.3]Z...[  
0070 2d 60 2a d6 04 b5 81 5d 81 5a c8 94 b5 81 5d 81  
0080 2d 1c 0a d6 02 ad 64 52 91 ea 70 7a 57 44 81 f0  
0090 c9 f1 0f 9f 7c e8 34 39 78 f1 8e f1 b3 c6 76 be  
00a0 9f c3 cc 48 e5 2d ff 00 5a 13 33 5d 5a e1 8a 5b

### El segmento TCP tiene un tamaño de: 1031 octetos



# EJEMPLO:

## ANÁLISIS DE LA DESCARGA POR “TROZOS” DE UNA IMAGEN

The image shows a Wireshark packet capture of an HTTP download. The packet list shows a sequence of TCP segments from 191.236.16.125 to 10.107.21.103, followed by an HTTP 200 OK response (packet 168) which is a JPEG image. The packet details pane for packet 168 shows the HTTP response structure, including the 'Data chunk (1024 octets)' section. The 'Chunk size: 1024 octets' is highlighted with a red box and labeled 'Tamaño del fragmento en notación decimal'. The 'Chunk boundary: 0d0a' is also highlighted with a red box and labeled 'Tamaño del fragmento en notación hexadecimal'. The packet bytes pane shows the raw data, with the first few bytes '34 30 30 0d 0a ff d8 ff e1 00 18 45 78 69' highlighted with a red box and labeled '400...'. The packet bytes pane also shows the 'De-chunked entity body (33653 bytes)'.

No.	Time	Source	Destination	Protocol	Length	Info
146	7.557819	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
147	7.557953	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [ACK] Seq=155 Ack=28144 Win=66048 Len=0
149	7.671041	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
151	7.778514	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
152	7.778684	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [ACK] Seq=155 Ack=30206 Win=66048 Len=0
155	7.888757	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
159	7.997952	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
160	7.998113	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [ACK] Seq=155 Ack=32268 Win=66048 Len=0
162	8.106597	191.236.16.125	10.107.21.103	TCP	1085	[TCP segment of a reassembled PDU]
164	8.218320	191.236.16.125	10.107.21.103	TCP	946	[TCP segment of a reassembled PDU]
165	8.218500	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [ACK] Seq=155 Ack=34191 Win=66048 Len=0
168	8.324934	191.236.16.125	10.107.21.103	HTTP	60	HTTP/1.1 200 OK (JPEG JFIF image)
170	8.355338	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [FIN, ACK] Seq=155 Ack=34196 Win=66048 Len=0
172	8.483130	191.236.16.125	10.107.21.103	TCP	60	80 → 50021 [FIN, ACK] Seq=34196 Ack=156 Win=131328 Len=0
173	8.483458	10.107.21.103	191.236.16.125	TCP	54	50021 → 80 [ACK] Seq=156 Ack=34197 Win=66048 Len=0

HTTP chunked response

- Data chunk (1024 octets)
  - Chunk size: 1024 octets
  - Data (1024 bytes)
  - Chunk boundary: 0d0a
- Data chunk (1024 octets)
  - Chunk size: 1024 octets
  - Data (1024 bytes)
  - Chunk boundary: 0d0a
- Data chunk (1024 octets)
  - Chunk size: 1024 octets

Longitud de los fragmentos de mensajes HTTP: 1024 octavas

Frame (60 bytes) Reassembled TCP (34195 bytes) De-chunked entity body (33653 bytes)

# EJEMPLO:

## ANÁLISIS DE LA DESCARGA POR “TROZOS” DE UNA IMAGEN

- RESUMEN:
  - Como se puede ver en las imágenes anteriores,
    - La longitud del contenido del segmento TCP es: **1031** octetos
    - La longitud del trozo del mensaje HTTP es: **1024** octetos
  - Un fragmento HTTP tiene la siguiente forma:
    - `LONG_FRAGMENTO\r\nCONTENIDO_FRAGMENTO\r\n`
  - El fragmento HTTP se mete en el campo de contenido de TCP, por tanto:
    - $\text{len}("1024") + \text{len}("\r\n") + 1024 + \text{len}("\r\n") = 4 + 2 + 1024 + 2 = \mathbf{1032}$
  - $1031 \neq 1032 \rightarrow$  ¿Por qué no coinciden los números?
    - ¡¡¡La longitud del fragmento debe meterse en notación hexadecimal!!!
      - 1024 en notación decimal = 400 en notación hexadecimal
      - $\text{len}("400") + \text{len}("\r\n") + 1024 + \text{len}("\r\n") = 3 + 2 + 1024 + 2 = \mathbf{1031}$

# FUNCIONAMIENTO DE HTTP:

## CUERPO DEL MENSAJE

---

- En el cuerpo del mensaje se envían octetos (bytes) que pueden representar:
  - Texto (texto plano, HTML, CSS, XML, JSON, CSS, ...)
  - Contenido binario (PDF, vídeo, ejecutables, etc.)
- El contenido binario está destinado a aplicaciones o plataformas particulares. Por ejemplo, los PDF solo pueden ser entendidos por un lector de PDF. Si se abre un PDF con un editor de texto plano o se envía a una salida estándar, se muestra una representación incorrecta de su contenido, con base de texto.
- El texto puede ser visualizado en multitud de programas, pero no todos usan la misma configuración para decodificar los octetos.
  - Por ejemplo, si un fichero de texto fue codificado originalmente en [UTF-8](#), el símbolo "ñ" se grabará como C3 B1.
  - Si intentamos abrir el mismo fichero con un editor de textos configurado en [ISO-8859-1](#), codificación en la que cada carácter se codifica con un solo octeto, la decodificación dará como resultado dos símbolos: Ã±.

# FUNCIONAMIENTO DE HTTP:

## ACLARACIONES SOBRE CODIFICACIÓN

Código	octeto 4	octeto 3	octeto 2	octeto 1
<b>US-ASCII (7 bits)</b>				
<b>ISO-8859-1 (1 octeto)</b>				
<b>UTF-8 (1 a 4 octetos)</b>				0
			1 1 0	1 0
		1 1 1 0	1 0	1 0
	1 1 1 1 0	1 0	1 0	1 0

- Latin-1 (ISO-8879-1) y UTF-8 llevan dentro la tabla del código US-ASCII.
- En latin-1 los símbolos añadidos a US-ASCII tiene n “1” en el bit mas significativo (MSB - Most Significant Bit).
- En UTF-8 los símbolos añadidos a US-ASCII por Latin-1 se codifican con dos octetos.

# FUNCIONAMIENTO DE HTTP:

## ACLARACIONES SOBRE CODIFICACIÓN

Código	Carácter Gráfico	Hex		binario																							
US-ASCII (7 bits)	Z	--	5A										1	0	1	1	0	1	0								
	ñ	--	--																								
	á	--	--																								
ISO-8869-1 (1 octeto)	Z	--	5A									0	1	0	1	1	0	1	0								
	ñ	--	F1									1	1	1	1	0	0	0	1								
	á	--	E1									1	1	1	0	0	0	0	1								
UTF 8 ( 1- 4 octetos)	Z	--	5A									0	1	0	1	1	0	1	0								
	ñ	C3	B1									1	1	0	0	0	0	1	1	1	0	1	1	0	0	0	1
	á	C3	A1									1	1	0	0	0	0	1	1	1	0	1	0	0	0	0	1

\* Supongamos que se desea codificar el siguiente texto: Hola Iñaki Pérez!

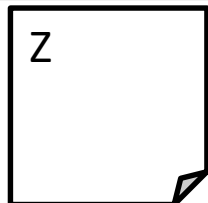
US-ASCII: No se puede codificar, porque los símbolos “ñ” y “é” no están recogidos en el alfabeto US-ASCII.

Latin-1: 48 6f 6c 61 20 49 f1 61 6b 69 20 50 e9 72 65 7a 21

UTF-8: 48 6f 6c 61 20 49 c3 b1 61 6b 69 20 50 c3 a9 72 65 7a 21

# FUNCIONAMIENTO DE HTTP:

## ACLARACIONES SOBRE CODIFICACIÓN



**guardar**

ASCII

Latin-1

UTF-8

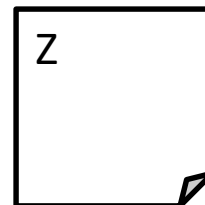
5A

**abrir**

ASCII

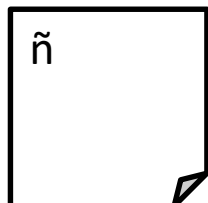
Latin-1

UTF-8



**guardar**

ASCII



**guardar**

Latin-1

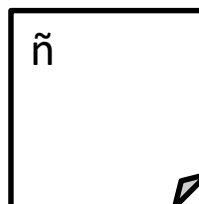
F1

**abrir**

ASCII

Latin-1

UTF-8



**guardar**

UTF-8

C3 B1

**abrir**

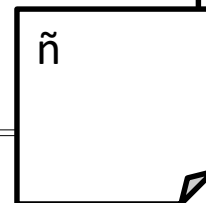
ASCII

Latin-1

UTF-8



Ã ±



# FUNCIONAMIENTO DE HTTP:

## CUERPO DEL MENSAJE

---

- El servidor web puede indicar a través de la cabecera “Content-Type” el tipo de contenido que está devolviendo: text/html, application/pdf, ...
- Cuando el navegador lee la cabecera “Content-Type”, comprueba si su valor es de tipo texto o de otro tipo.
  - Si el contenido es de otro tipo, comprueba si tiene algún plugin configurado para abrirlo. Si no lo tiene, ofrece al usuario la opción de guardar el contenido o abrirlo con la aplicación correspondiente.
  - Si el contenido es de tipo texto, él mismo trata de interpretarlo (visualizarlo). Para ello, necesita saber cómo está codificado\*.
    - HTTP permite especificar la codificación del contenido en la cabecera “Content-Type”:

Content-Type: text/html; **charset=ISO-8859-1 (Latin-1)**

# EJEMPLO:

## ANÁLISIS DE UN PROBLEMA DE CODIFICACIÓN

Usando Wireshark, abre la siguiente URI en un navegador y encuentre la causa de su problema de codificación

<http://ws-responsecontentcoding.appspot.com/>





# EJEMPLO:

## ANÁLISIS DE UN PROBLEMA DE CODIFICACIÓN

\*Conexión de red inalámbrica

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
535	2...	10.107.26.226	66.102.1.141	HTTP	398	GET / HTTP/1.1
539	2...	66.102.1.141	10.107.26.226	HTTP	309	HTTP/1.1 200 OK (text/plain)

Frame 539: 309 bytes on wire (2472 bits), 309 bytes captured (2472 bits) on interface 0

Ethernet II, Src: CheckPoi\_3a:b1:d9 (00:1c:7f:3a:b1:d9), Dst: IntelCor\_f2:d2:5c (10:0b:a9:f2:d2:5c)

Internet Protocol Version 4, Src: 66.102.1.141, Dst: 10.107.26.226

Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49511 (49511), Seq: 1, Ack: 346, Len: 255

Hypertext Transfer Protocol

HTTP/1.1 200 OK\r\n

Cache-Control: no-cache\r\n

Content-Type: text/plain; charset=latin-1\r\n

Content-Encoding: gzip\r\n

Vary: Accept-Encoding\r\n

Date: Thu, 04 Feb 2016 08:25:33 GMT\r\n

Server: Google Frontend\r\n

Content-Length: 39\r\n\r\n

[HTTP response 1/1]

[Time since request: 0.147155000 seconds]

[Request in frame: 535]

Content-encoded entity body (gzip): 39 bytes -> 19 bytes

Line-based text data: text/plain

Hola I\303\261aki P\303\251rez!

0000 48 6f 6c 61 20 49 c3 b1 61 6b 69 20 50 c3 a9 72 Hola I.. aki P..r

0010 65 7a 21 ez!

Frame (309)

Text item (text), 19 bytes

Packets: 646 • Displayed: 2 (0.3%) Profile: Default

Contenido en notación  
hexadecimal

- El navegador ve el contenido como "text/plain", por lo que es responsable de interpretar el contenido.
- ¿Cómo debe ser interpretado el texto por el navegador? Es decir, ¿qué tabla de codificación debería usar? Como el parámetro "charset" indica; en este caso: latin-1.

48 → H

6F → o

6C → l

61 → a

20 →

49 → I

C3 → Ñ

B1 → ±

# EJEMPLO:

## ANÁLISIS DE UN PROBLEMA DE CODIFICACIÓN

- ¿Por qué ocurre esto?
  - Porque quien ha programado el servidor ha escrito una línea que indica que el contenido está codificado en latin-1:

```
self.response.charset = 'latin-1'
```

- pero no ha tenido en cuenta que el servidor, cuando utiliza el método *write*, codifica el contenido en UTF-8

```
self.response.write("Hola Iñaki Pérez!")
```

Content-Type: text/plain; charset: latin-1

```
import webapp2
class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.content_type = 'text/plain'
        self.response.charset = 'latin-1'
        self.response.write("Hola Iñaki Pérez!")

app = webapp2.WSGIApplication([('/', MainHandler)], debug=True)
```

# FUNCIONAMIENTO DE HTTP:

## COMPRESIÓN

---

- ¿Cómo se reduce el tamaño de las respuestas?

- En general, el contenido de una respuesta HTTP es texto: HTML, XML, JSON, CSS.
- El texto tiene buena respuesta a la compresión.
- El protocolo HTTP permite comprimir el cuerpo del mensaje, siendo el algoritmo más utilizado para ello el *gzip*.
- El algoritmo de compresión se indica en dos cabeceras:
  - En la **solicitud**: “**Accept-Encoding**”
    - Si el cliente no tiene capacidad para descomprimir un contenido comprimido, este encabezamiento tomará el valor de "Identity".
  - En la **respuesta**: “**Content-Encoding**”
    - Si el cuerpo del mensaje se envía sin comprimir, esta cabecera no se debe introducir en la respuesta.

# FUNCIONAMIENTO DE HTTP:

## COMPRESIÓN

- Supongamos que un cliente, capaz de trabajar con contenidos comprimidos, solicita un recurso con URI : <http://www.google.es/>


### Solicitud de ejemplo

```
GET / HTTP/1.1
Host: www.google.es
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

### Respuesta de ejemplo

```
HTTP/1.1 200 OK
Date: Mon, 30 Nov 2015 21:03:14 GMT
Content-Encoding: gzip
Content-Length: 5543
Content-Type: text/html; charset=ISO-8859-1
```

CONTENIDO COMPRIMIDO



```
0Ú|ñ"â¥ÊîF|@eL-Ñ6|æYÁ|HÀ?e'S|Ö■rÜ_Q1|EF±E|_#'^-|
|e%ôE|l■B
8Ó_T0üs°L18UâS y ÆX"|B*-i5]²Äü":â%Ç_Tô¶e*M-:óA(=ô!|Ä²X#çLd||ð±`
²@70Ç ä      ²²ád0Tê' LH?NÇx2VâÉ£i4B°Ç-âMÇæ²Ai2É-üy0a:8îâÇAÎ
(â²)²æDr*-B!±ú=E-âc²¶UÐÉ_TÇi||c 4$F|â||ô:h'1Séu'=ÇÖbätüüYüB||
```

# FUNCIONAMIENTO DE HTTP:

## COMPRESIÓN - EJEMPLO

```
import requests
import sys
import zlib

metodo = 'GET'
uri = "https://www.google.es/"
cabeceras = {'Host': 'www.google.es'}

compressed = False
if len(sys.argv) == 1:
    cabeceras['Accept-Encoding'] = 'identity'
elif sys.argv[1] == 'compress':
    compressed = True
    cabeceras['Accept-Encoding'] = 'gzip'
else:
    print("Error! Erabilera: python compression_es.py compress")
    exit(0)

respuesta = requests.request(metodo, uri, headers=cabeceras, allow_redirects=False, stream=True)

codigo = respuesta.status_code
descripcion = respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])

print("RESPONSE CONTENT LENGTH: " + str(len(respuesta.raw.data)) + " byte")
if compressed:
    contenido_compressed = respuesta.raw.data
    contenido_uncompressed = zlib.decompress(contenido_compressed, 16+zlib.MAX_WBITS)
    print("UNCOMPRESSED RESPONSE CONTENT LENGTH: " + str(len(contenido_uncompressed)) + " byte")
```

- La librería requests descomprime automáticamente los formatos de compresión gzip y deflate.
- Si queremos un acceso directo al contenido de la respuesta, introducimos el parámetro `stream = True` en la solicitud y leemos el campo `raw.data` en la respuesta

```
graph TD
    A[stream=True] --> B[respuesta.raw.data]
    B --> C[contenido_compressed.raw.data]
```

# FUNCIONAMIENTO DE HTTP:

## COMPRESIÓN - EJEMPLO

```
(venv) C:\Users\cvzcaio\Dropbox (Personal)\docencia\grado\Sistemas Web\SW 2020\Praktikal>python compression.py compress 200 OK
Date: Wed, 12 Feb 2020 07:39:19 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Content-Encoding: gzip
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-02-12-07; expires=Fri, 13-Mar-2020 07:39:19 GMT; path=/; domain=.google.es; Secure, NID=197=TuCqh
sRxyTUWCZOVawM6ScZOBD0; expires=Thu, 13-Aug-2020 07:39:19 GMT; path=/; domain=.google.es; HttpOnly
Alt-Svc: quic=":443"; ma=2592000; v="46,43",h3-Q050=":443"; ma=2592000,h3-Q049=":443"; ma=2592000,h3-Q048=":443"; ma=2592000
Transfer-Encoding: chunked
RESPONSE CONTENT LENGTH: 5584 byte
UNCOMPRESSED RESPONSE CONTENT LENGTH: 13116 byte
```

- Tamaño de la respuesta comprimida: 5584 octetos
  - Tamaño de la respuesta sin comprimir: 13116 octetos
- Un 57% de ahorro en el tamaño del cuerpo del mensaje