Engineer

# UTF-8

Codificación Unicode de uno a cuatro bytes por valor escalar

## Qué es

UTF-8 es la codificación que convierte el texto Unicode en bytes. El estándar Unicode la define como la forma de codificación que guarda cada valor escalar Unicode, es decir, cualquier punto de código salvo los sustitutos reservados para UTF-16, en una secuencia de uno a cuatro bytes. Un carácter percibido por el usuario puede contener varios valores escalares y necesitar, por tanto, más bytes. Es una de las tres formas de codificación del estándar, junto con UTF-16 y UTF-32, y la que exigen los estándares de la propia web. Como etiqueta, en una página HTML o una cabecera HTTP, suele escribirse en minúsculas, utf-8, y a veces sin guion, utf8; el estándar Encoding acepta ambas y no distingue mayúsculas de minúsculas en las etiquetas.

Ken Thompson diseñó UTF-8 en septiembre de 1992 para el sistema operativo Plan 9, siguiendo los criterios que había establecido Rob Pike. Durante su estandarización, la codificación se llamó FSS-UTF y después UTF-2, antes de recibir su nombre actual. RFC 3629, la definición del IETF de 2003, la limita a los puntos de código U+0000 a U+10FFFF, el mismo espacio de códigos de Unicode. Cuando se escribió el RFC, la descripción de UTF-8 en ISO/IEC 10646 aún permitía secuencias de hasta seis bytes; según el RFC y el estándar Unicode, ningún valor escalar Unicode ocupa más de cuatro.

Los bits iniciales de cada byte indican qué tipo de byte es. Un byte cuyo bit más significativo es 0 representa un carácter por sí solo: uno de los 128 caracteres de ASCII. Un byte que empieza por 110, 1110 o 11110 inicia una secuencia de dos, tres o cuatro bytes; cada byte posterior empieza por 10 y lleva otros seis bits del punto de código. El formato completo cabe en cuatro filas, donde x representa los bits del punto de código:

```
U+0000..U+007F     0xxxxxxx
U+0080..U+07FF     110xxxxx 10xxxxxx
U+0800..U+FFFF     1110xxxx 10xxxxxx 10xxxxxx
U+10000..U+10FFFF  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

La fila de tres bytes excluye los sustitutos, U+D800 a U+DFFF: no son caracteres y cualquier secuencia de bytes que codifique uno está mal formada.

De esa estructura se derivan cuatro propiedades, todas descritas por el estándar Unicode. UTF-8 es transparente para ASCII: los bytes 00 a 7F conservan su significado de ASCII y nunca aparecen dentro de otro carácter. Es autosincronizable: resulta fácil distinguir un byte inicial de uno de continuación. Ordenar cadenas UTF-8 byte a byte produce el mismo orden que ordenarlas por punto de código. Y no tiene orden de bytes, por lo que nunca surge la cuestión del orden big-endian o little-endian que deben resolver UTF-16 y UTF-32.

Utilice el concepto en una decisión

Aplique «¿Qué es UTF-8?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Software que se lee con naturalidad en cada mercado donde vende, incluidas las fechas, las monedas y la forma de ordenar los nombres.

**Explore nuestros servicios** Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. i18n en inglés y danés — 12,027 mensajes por configuración regional en inglés y danés

**i18n en inglés y danés** Póngase en contacto →

## Por qué importa

La web lo ha adoptado. El estándar WHATWG Encoding, la especificación que implementan los navegadores para leer y escribir texto, identifica UTF-8 como la forma preferida de intercambiar texto Unicode y lo exige para protocolos y formatos nuevos, así como para formatos existentes usados en contextos nuevos. Indica a los autores que utilicen UTF-8 y lo identifiquen con la etiqueta utf-8. HTML sigue el mismo criterio: la codificación de un documento debe ser UTF-8, lo declare o no, y las herramientas de creación deben usar UTF-8 como opción predeterminada para documentos nuevos. Ya se espera que un producto que publique páginas, envíe JSON o acepte formularios enviados por POST lo utilice.

Su diseño explica por qué se consolidó esa elección. Como los bytes ASCII nunca aparecen dentro de otro carácter, un programa escrito para ASCII sigue funcionando con texto UTF-8: un analizador que separe por comas, barras o saltos de línea los encuentra exactamente donde están en una frase danesa o ucraniana, y nunca en medio de una å o una Ґ. Como un programa situado en cualquier posición encuentra el inicio de un valor escalar codificado buscando como máximo cuatro bytes hacia atrás, y normalmente menos, la decodificación puede reanudarse en una secuencia posterior bien formada, en lugar de perder el resto del archivo. Eso no garantiza exactamente un carácter de sustitución por cada byte dañado. Buscar un valor escalar codificado completo nunca coincide con parte de otro. Como el orden binario es el de los puntos de código, una base de datos o un programa puede ordenar bytes UTF-8 sin decodificarlos. Ese orden es técnico: coloca å antes de æ y ø, mientras que un diccionario danés termina el alfabeto con æ, ø y å.

El coste está en el tamaño. El estándar Unicode considera UTF-8 razonablemente compacto: la sintaxis ASCII y los idiomas occidentales ocupan mucho menos que con UTF-16, mientras que el texto en sistemas de escritura asiáticos como los del hindi, tailandés, chino, japonés y coreano ocupa bastante más. En el alfabeto danés básico, las letras ocupan un byte salvo æ, ø y å, que ocupan dos; otras letras acentuadas pueden necesitar más. El ucraniano necesita dos bytes por cada letra cirílica y uno por los espacios, dígitos y signos de puntuación ASCII, pero otra puntuación puede ocupar más. Por tanto, un campo medido en bytes admite aproximadamente la mitad de letras ucranianas que inglesas.

## Cómo se aplica en la práctica

Los bytes se ven con facilidad. Node.js tiene, igual que un navegador, el `TextEncoder` del estándar Encoding, que escribe una cadena como UTF-8:

```
const hex = (bytes) =>
  [...bytes].map((b) => b.toString(16).padStart(2, "0")).join(" ");
const encoder = new TextEncoder();
for (const text of ["A", "å", "Ґ", "€", "𐍈"]) {
  console.log(text, hex(encoder.encode(text)).toUpperCase());
}
```

Ejecutado en Node.js 25, imprime cada carácter y sus bytes en hexadecimal:

```
A 41
å C3 A5
Ґ D2 90
€ E2 82 AC
𐌂 F0 90 8C 82
```

La letra A es ASCII y conserva su único byte. La å danesa es U+00E5, que necesita ocho bits, uno más que los siete que ofrece un solo byte, de modo que usa el patrón de dos bytes. Su valor de once bits, 00011100101, se divide en 00011 y 100101; el byte inicial es 110 seguido de 00011, que da C3, y el de continuación, 10 seguido de 100101, que da A5. La Ґ ucraniana, U+0490, ocupa dos bytes del mismo modo; el símbolo del euro ocupa tres y la letra del antiguo alfabeto itálico 𐌂, situada más allá de U+FFFF, cuatro. El primer byte de cada secuencia anuncia su longitud: C3 y D2 empiezan por 110, E2 por 1110 y F0 por 11110.

Este sitio escribe todas las páginas en UTF-8 y lo declara donde exige HTML. El primer elemento de la cabecera de cada página es `<meta charset="utf-8">`, con la etiqueta utf-8 que prescribe el estándar Encoding y claramente dentro de los primeros 1.024 bytes, donde HTML exige que figure. La página de créditos tiene una sección de comprobaciones externas, ninguna propia: cada una ejecuta una prueba en directo, en lugar de aceptar sin más lo que afirmamos. Una de ellas es W3C Internationalization Checker, que lee las declaraciones de idioma y codificación que permiten que cada versión lingüística de este sitio resulte legible para su público.

## Dónde surgen los problemas

**Un límite de bytes corta un carácter por la mitad.** Una columna, un campo de mensaje o un búfer medido en bytes puede terminar en medio de una secuencia. El nombre Ærø tiene tres caracteres y cinco bytes; un límite de cuatro conserva Æ, r y el primer byte de ø:

```
const bytes = new TextEncoder().encode("Ærø");
const cut = new TextDecoder().decode(bytes.slice(0, 4));
console.log(bytes.length, [...cut].map((c) => c.codePointAt(0).toString(16)));
console.log(new TextEncoder().encodeInto("Ærø", new Uint8Array(4)));
```

```
5 [ 'c6', '72', 'fffd' ]
{ read: 2, written: 3 }
```

El byte inicial huérfano se decodifica como U+FFFD, el carácter de sustitución, por lo que el nombre guardado termina con un marcador donde estaba la ø. Corte en un límite de valor escalar Unicode. `encodeInto` , en la última línea, lo hace por sí mismo: con espacio para cuatro bytes, solo escribe los tres de Æ y r, e indica en `read` cuánto tomó de la cadena, contado en unidades UTF-16. El código que recorta bytes que ya tiene puede retroceder sobre los bytes de continuación, que empiezan por 10, hasta el byte inicial, y cortar allí. Eso conserva UTF-8 bien formado; para preservar además un carácter completo tal como lo percibe el usuario, también hay que respetar los límites de los grupos de grafemas.

**El archivo comienza con una marca de orden de bytes.** UTF-8 no tiene orden de bytes y no necesita marca, pero un conversor que transforma UTF-16 con marca de orden de bytes en UTF-8 convierte la marca en los tres bytes EF BB BF. El estándar Unicode no los exige, dice que su presencia no afecta a la conformidad y no los recomienda como firma de UTF-8. Los lectores discrepan en cómo tratarlos. Aquí, el archivo bom.json contiene esos tres bytes seguidos de `{"name":"Ærø"}` :

```
import { readFileSync } from "node:fs";

const text = readFileSync("bom.json", "utf8");
console.log(text.codePointAt(0).toString(16));
try {
  JSON.parse(text);
} catch (error) {
  console.log(error.name);
}
console.log(JSON.parse(new TextDecoder().decode(readFileSync("bom.json"))));
```

```
feff
SyntaxError
{ name: 'Ærø' }
```

El `readFileSync` de Node conserva la marca como el carácter U+FEFF y `JSON.parse` rechaza el texto por ese motivo. `TextDecoder` sigue el estándar Encoding, retira la marca y permite analizar correctamente los mismos bytes. En un navegador, la marca tiene aún más alcance: el estándar Encoding le da prioridad sobre cualquier otra indicación, incluida la cabecera HTTP `Content-`

`Type` . Escriba UTF-8 sin marca y, cuando lleguen archivos de otros sistemas, decodifíquelos con un decodificador que la retire, o elimine un U+FEFF inicial antes del análisis.

**Una comprobación lee los bytes antes de decodificarlos.** RFC 3629 describe un analizador que comprueba la entrada codificada y rechaza los bytes 2F 2E 2E 2F, el paso de ruta “/../”, pero deja pasar 2F C0 AE 2E 2F, donde C0 AE es una representación ilegal de dos bytes de un punto, que el analizador acaba interpretando como un único punto. El RFC registra que un virus que se propagó ampliamente por servidores web en 2001 usó precisamente este truco. Desde Unicode 3.1, estas formas que no usan la representación más corta están mal formadas, y los bytes C0, C1 y F5 a FF nunca aparecen en UTF-8 válido. Un decodificador conforme, incluido `TextDecoder` , transforma esos cinco bytes en una barra, dos U+FFFD y “./”, por lo que ya no queda ningún “/../” que pueda burlar la comprobación. El estándar Unicode es igual de estricto con lo que sigue a un error: un decodificador que se traga los bytes válidos posteriores a un byte inicial incorrecto no es conforme y permite ataques. `TextDecoder` convierte los bytes C2 41 42 en U+FFFD seguido de AB. Decodifique primero con el decodificador de la propia plataforma; después compruebe el texto decodificado. Nunca escriba a mano un decodificador UTF-8 ni un patrón de bytes que lo sustituya.

**La sustitución oculta el daño.** Un decodificador sustituye de forma predeterminada cada secuencia incorrecta por U+FFFD y continúa. Eso es adecuado para mostrar una página, pero no para importar datos: el texto parece casi normal, la copia guardada ha perdido los bytes originales y quien lo envió nunca descubre dónde se dañó el archivo. El estándar Encoding define dos modos de error para decodificar, «replacement» y «fatal»; para identificadores y secuencias de bytes dentro de un formato o protocolo, deja que otros estándares elijan un decodificador que falle en lugar de sustituir. En JavaScript, `new TextDecoder("utf-8", { fatal: true })` lanza un `TypeError` para el nombre cortado del ejemplo, en lugar de devolverlo. Use el modo fatal siempre que el texto se almacene, se firme o se compare; rechace la entrada con un mensaje que permita corregir el problema y reserve la sustitución para la visualización.

**La declaración llega tarde, falta o es incorrecta.** HTML exige que el elemento que declara la codificación esté completo dentro de los primeros 1.024 bytes del documento. Exige una declaración incluso si todos los caracteres son ASCII, porque el navegador necesita una codificación para lo que se escribe en los formularios y para las URL que construyen los scripts. Una etiqueta que parece inocua también puede ser incorrecta: en el estándar Encoding, “ascii”, “latin1” e

“iso-8859-1” son etiquetas de windows-1252, por lo que una página declarada como ASCII se lee como windows-1252. Ponga `<meta charset="utf-8">` al principio de la cabecera, envíe `Content-Type: text/html; charset=utf-8` con cada página y considere un defecto cualquier otra etiqueta en cualquiera de esos lugares.

## Lecturas adicionales

- The Unicode Standard, capítulo 3: Conformance — la definición de UTF-8, la tabla de secuencias de bytes bien formadas, la marca de orden de bytes y cómo debe tratar un conversor la entrada mal formada.
- The Unicode Standard, capítulo 2: General Structure — por qué UTF-8 es transparente para ASCII, autosincronizable y carece de orden de bytes, y cómo se compara su tamaño con UTF-16.
- RFC 3629: UTF-8, a transformation format of ISO 10646 — la definición del IETF, su historia desde Plan 9 y las consideraciones de seguridad del ataque con una codificación no mínima de la ruta.
- Encoding Standard — la especificación WHATWG que siguen los navegadores: por qué se exige UTF-8, cómo trata la decodificación la marca de orden de bytes y los errores, y las interfaces `TextEncoder` y `TextDecoder`.
- HTML: Specifying the document’s character encoding — qué debe decir una declaración, dónde debe aparecer y por qué la necesita incluso una página escrita solo en ASCII.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

**Vea cuánto cuesta** Póngase en contacto →

# Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es UTF-8?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

**Explore nuestros servicios**

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/utf-8/