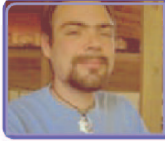


# Codificación de Caracteres

ENTENDIENDO EL POR QUÉ DE UNICODE



**Gunnar Wolf** es administrador de sistemas para el Instituto de Investigaciones Económicas de la UNAM; entusiasta y promotor del Software Libre, desarrollador del proyecto Debian GNU/Linux desde el 2003, miembro externo del Departamento de Seguridad en Cómputo de DGSCA-UNAM desde 1999.

Hace un par de días recibí uno de esos correos que parecen venir de una época ya superada y olvidada hace años. Dicho correo comenzaba con la frase:

*Para evitar problemas de compatibilidad, este correo no incluye acentos ni enies.*

En efecto, este problema casi ha desaparecido del correo, y ese pretexto sencillamente ya no resulta ni siquiera una excusa adecuada para quien le da flojera escribir el español correctamente. Sin embargo, no en todos los campos podemos hablar de un éxito tan grande como en el manejo del correo electrónico. Si bien para los hispanoparlantes el correo funcionó casi bien desde un principio, cuando el uso de Internet dio su gran salto cuantitativo en los 90s, para los nativos de muchas culturas alrededor del mundo se hizo imperativo encontrar cómo comunicarse confiablemente.

Pero el problema viene de mucho más atrás. Repasemos rápidamente la historia de los esquemas de codificación de caracteres.

## Repaso histórico

La codificación a partir de la cual se originan todos los esquemas en uso hoy en día nació en 1963, revisado/ratificado en 1967 con el nombre de ASCII (Código Estándar Americano para el Intercambio de Información). ASCII es un código de 7 bits, permitiendo la representación de hasta 128 caracteres: 32 caracteres de control, 34 símbolos, 52 caracteres de texto (mayúsculas y minúsculas) y 10 dígitos.

Sobra decir que el ámbito del cómputo ha cambiado drásticamente desde 1963. La computadora debía representar apenas la información indispensable para ser comprendida por sus operadores, y en la propuesta original, ASCII no incluía ni siquiera letras minúsculas. Pero ya en 1964 aparecieron las máquinas de escribir con la capacidad de guardar (y corregir) páginas en cinta magnética. Fue sólo cuestión de tiempo para que estas capacidades quedaran al alcance de todo mundo.

Todos los idiomas europeos que utilizan el alfabeto latino, a excepción del inglés, requieren de diferentes tipos de signos diacríticos. Para acomodar esto, hacia fines de los 70s todas las computadoras ampliamente desplegadas habían estandarizado en un tamaño de palabra de 8 bits, con lo que se creó un ASCII ampliado que daría

128 caracteres adicionales. Sin embargo, no surgió un estándar para su uso. Además, muchos de estos caracteres fueron empleados para incluir caracteres semigráficos que permitieran construir interfaces amigables al usuario.

En 1981, IBM puso a la venta su IBM PC. Contaba con una tarjeta de video con páginas de códigos reprogramables. El espacio de caracteres podía ser redefinido por software; los caracteres cargados por omisión se denominaron página de códigos 437 (CP437), dando soporte parcial para algunos lenguajes europeos. Pero debido a los caracteres semigráficos este espacio no era suficiente, por lo que era necesario activar una página de código alternativa —para el español, la CP850. La situación mejoró al popularizarse los entornos gráficos y dejar de depender de éstos caracteres especiales. Entonces, las hojas de código similares pudieron agruparse y fue así que nacieron conjuntos de caracteres como el ISO-8859-1 para los lenguajes occidentales.

El problema se presenta al intercambiar archivos con usuarios de otras páginas: los datos que usan una página son indistinguibles de los que usan otra. Si compartieras un archivo ISO-8859-1 con una persona de Europa oriental (ISO-8859-2), los caracteres acentuados aparecerían modificados, aparentemente corruptos. La situación de los lenguajes de Asia oriental era mucho peor aún, dado que plantear el uso de un alfabeto de 256 caracteres resultó imposible, y estos países desarrollaron esquemas paralelos de representación.

En 1988, desarrolladores de Xerox y Apple, se reunieron para atacar este problema de raíz. Lanzaron la iniciativa del sistema Unicode, buscando aprovechar los grandes avances de más de 20 años del cómputo para lograr un conjunto de caracteres apto para todo el mundo. Pronto su iniciativa logró captar la atención y el respaldo de otros líderes. Para 1996 se publicó la especificación Unicode 2.0, permitiendo un espacio de más de un millón de puntos de código independientes, reteniendo compatibilidad hacia atrás completa con ISO-8859-1.

Unicode es tan grande que su representación interna no está libre de polémica. Sin entrar en detalles técnicos, las dos principales representaciones son UTF-8 (utilizada en sistemas basados en Unix, y en general, para toda transmisión sobre redes de datos) y UTF-16 (en uso principalmente en sistemas Windows). UTF-8 está

## “Unicode es tan grande que su representación interna no está libre de polémica”.

basado en elementos individuales de 8 bits, mientras que el átomo en UTF-16 es de 16 bits, por lo que típicamente UTF-8 es más compacto y por lo tanto más robusto para transmisiones sobre la red. Un punto importante a tener en cuenta es que la representación binaria de texto ASCII heredado es idéntica a su representación en UTF-8, pero no así con UTF-16. Para más detalles respecto a diferencias entre estas dos representaciones, ver “Wikipedia Comparación” en la lista de referencias.


### ¿Por qué es diferente el manejo de Unicode?

Hasta antes de Unicode, todo lo que teníamos que saber respecto a un esquema de codificación se limitaba a elegir la salida adecuada, o interpretar la entrada como debía ser. Pero Unicode tiene muchas particularidades que requieren de un manejo especial. Algunos de ellos, son:

- **Longitud de una cadena.** Como programadores estamos acostumbrados a que la longitud en caracteres de una cadena sea igual a su tamaño en bytes; Unicode rompe con este supuesto. Para medir la longitud de una cadena Unicode tenemos que evaluar la cadena completa y encontrar cuáles partículas son de qué tamaño, ya que un carácter puede medir entre uno y seis bytes. Además hay caracteres “combinantes”, que no ocupan un espacio por sí solos sino que modifican el carácter anterior.
- **No todas las cadenas son válidas.** Con los esquemas tradicionales de 8 bits, todo conjunto de caracteres es válido. Al hablar de las representaciones de Unicode, hay cadenas que resultan inválidas. Si has visto una página Web donde los caracteres con diacríticos aparecen substituídos por caracteres en forma de rombo con un signo de interrogación dentro (◊), éste carácter denota que el procesamiento de Unicode no pudo completarse. Sin embargo, no todas las aplicaciones son tan benignas como un navegador — Una base de datos debe negarse a guardar datos mal-formados — por lo cual debemos estar conscientes del tipo de excepciones que recibiremos si nuestro usuario nos da datos inválidos.
- **Caracteres definidos semánticamente, no visualmente.** Los caracteres en Unicode no buscan únicamente representar un grupo de grafías, sino que su significado. En tiempos del ASCII nos acostumbramos a utilizar la representación gráfica más cercana a un carácter. Por ejemplo, nos acostumbramos a representar a la multiplicación ya sea con la letra «x» o con el asterisco; con Unicode podemos usar –cual debe ser– el símbolo × (U+2715). Los alemanes y griegos ya no confunden la ß (beta, U+00DF)

con la ß (Eszett o S fuerte, U+00DF). Aunque esta capacidad de Unicode es una ventaja, también puede llevarnos a confusiones. Por ejemplo, si bien en ASCII con CodePage 850 sólo podemos representar a una letra acentuada de una manera (la letra á está en la posición 160), en Unicode puede representarse de esta misma forma, o como la combinación de una ‘a’ minúscula y un carácter combinante de acento agudo (U+0300): á. Esto permite la gran expresividad que requieren algunos alfabetos, pero nos obliga a considerar más casos especiales al enfrentarnos a Unicode desde el punto de vista del desarrollador de sistemas.

### Conclusión

Viendo sólomente estos puntos, puede que muchos de ustedes los vayan poniendo en la balanza, y les parezca que migrar a Unicode es demasiado difícil y no vale la pena. Sin embargo, sencillamente, no nos queda opción: el mundo entero va avanzando en ese sentido. Si bien tenemos la fortuna de ser nativos de una cultura que utiliza el alfabeto dominante en Internet, no podemos ignorar que formamos parte de un mundo plenamente globalizado, y que los días de vivir en nuestra isleta feliz de un sólo alfabeto han terminado. 

» Por Gunnar Wolf

### Referencias

- [1]. (Spolsky 2003) The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) <http://bit.ly/2WvgB8>
- [2]. (Jennings 1999-2004) Annotated history of character codes <http://bit.ly/14nllX>
- [3]. (Dürst 1997) The Properties and Promizes of UTF-8 <http://bit.ly/3RxG1T>
- [4]. (Czyborra 1995-2000) Unicode in the Unix Environment <http://bit.ly/2h4kDH>
- [5]. (Wood 1999-2009) Combining Diacritical Marks <http://bit.ly/3qqZ1A>
- [6]. (Wikipedia Comparación) Comparison of Unicode encodings <http://bit.ly/1Uot00>