

Unicode en Perl

Guía de supervivencia

Enrique Nell
Barcelona.pm

Junio de 2006

Juegos de caracteres: era post-ASCII

- Hay más de 5.000 lenguas en el mundo.
- Para representar texto en idiomas distintos del inglés se desarrollaron varios juegos de caracteres, como ISO 8859 (de 256 caracteres), con sus 16 variantes (ISO 8859-1 = Latin-1).
- Los idiomas asiáticos requieren códigos para miles de caracteres.

Unicode

- Estándar desarrollado para codificar en una sola base de datos unificada todos los sistemas de escritura del mundo y muchos otros símbolos.
- Los caracteres están divididos en bloques contiguos.
- Define sistemas de escritura (scripts).
- Define diversas propiedades (p. ej., *lowercase*, *punctuation*, etc.) de los caracteres.
- Define diversas operaciones (p. ej., convertir a mayúsculas, ordenar, comparar, etc.) para los caracteres.

Representación de caracteres

- Unicode asocia a cada carácter un valor numérico único, denominado **código de carácter** (en inglés, *code point* o *code position*) y un **nombre descriptivo**.
- Los códigos de carácter se representan en notación **hexadecimal**.
- Ejemplo: el carácter **A** se representa así:
“U+0041 LATIN CAPITAL LETTER A”

Representación de caracteres

- Unicode define caracteres compuestos, formados por un carácter base seguido de uno o más modificadores (p. ej., marcas diacríticas). Ejemplo:

LATIN CAPITAL LETTER A + COMBINING ACUTE ACCENT

- Para algunas combinaciones (como los caracteres extendidos de Latin-1) hay caracteres predefinidos como un solo código de carácter. Ejemplo:

LATIN CAPITAL LETTER A WITH ACUTE

- Se han definido varias formas de **normalización** para estandarizar representaciones al convertir distintas combinaciones de caracteres.

Formas de codificación de caracteres

UTF-8:

- Codificación multibyte de longitud variable que utiliza entre 1 y 6 bytes para representar cada carácter Unicode (de momento no se ocupan más de 4).
- Los caracteres más comunes se representan con un solo byte; para representar los menos frecuentes se utilizan varios bytes.
- Define códigos para todos los caracteres de Latin-1 en las mismas posiciones => una secuencia de datos que no incluya caracteres que requieran más de 1 byte es 100% compatible con Latin-1 => los programas que esperen datos Latin-1 podrán leerla correctamente.
- Si un documento contiene muchos caracteres asiáticos, ocupará más en UTF-8 que en UTF-16.

Formas de codificación de caracteres

UTF-16:

- Codificación de 16 bits de caracteres Unicode. Es el formato nativo de Unicode. Hasta U+FFFF, cada carácter se codifica como un solo valor de 16 bits. A partir de ese punto, los códigos de caracteres se representan como pares de valores de 16 bits.
- Hay dos variantes: big-endian y little-endian.
- Si un documento sólo contiene caracteres ASCII, al codificarse con UTF-16 ocupará el doble que el equivalente ASCII.

Formas de codificación de caracteres

- UTF-32
- UCS-2: codificación en 2 bytes de ISO 10646.
- UCS-4: codificación en 4 bytes de ISO 10646.

Implementación en Perl

- Primera versión de Perl recomendada para trabajar con Unicode: 5.8.0 (según Simon Cozens, la 5.8.2).
- En la versión 5.6 el pragma utf8 se utilizaba para declarar que las operaciones de un bloque o archivo utilizaban Unicode.
- Ahora esto lo indican los mismos datos. Este pragma sólo es necesario en un caso: para utilizar caracteres UTF-8 en nombres de identificadores, en literales de cadena y en expresiones regulares. Ejemplo:

```
use utf8;  
binmode STDOUT, ":utf8";  
print "ネルエンリケ翻訳家です\n";
```

Implementación en Perl

- Se introduce un indicador (flag) utf8 implícito en los escalares. Sólo se tratarán como caracteres Unicode los escalares marcados con este indicador, aunque sean datos UTF-8 válidos.
- Para ver si una cadena tiene el indicador activado:
Encode::is_utf8()
utf8::is_utf8() (del módulo utf8, no del pragma utf8)
Devel::Peek (función Dump)
- Perl lee como datos Unicode los scripts marcados con BOM o codificados como UTF-16 sin BOM.

Implementación en Perl

- La carpeta unicore (v5.8) [o unicode (v5.6)] de la biblioteca estándar contienen el estándar Unicode representado en varios archivos de texto manipulables por un programa.
- Se puede utilizar el módulo `Unicode::UCD` para consultar esta información.

Bytes o caracteres

- Perl permite trabajar con bytes y con cadenas de caracteres Unicode.
- Internamente, Perl utiliza el juego de caracteres de 8 bits nativo de la plataforma y también utiliza UTF-8 para codificar caracteres Unicode.
- En el futuro las operaciones utilizarán caracteres Unicode en lugar de bytes.

Bytes o caracteres: vía de migración provisional

- Siempre que puede, Perl trabaja con bytes (normalmente se utiliza ISO 8859-1; se puede utilizar otra codificación con el pragma **encoding**).
- Si puede determinar que los datos de entrada son caracteres Unicode (por ejemplo, se añade una capa de codificación de caracteres a un identificador de archivo o un programa contiene un literal de cadena Unicode), cambia a semántica de caracteres. Si no, utiliza semántica de bytes.
- Para interpretar cadenas de bytes como UTF-8: use encoding 'utf8';
- Para forzar la semántica de bytes en un ámbito léxico determinado: use bytes;

Entrada/Salida

- PerlIO: nuevo sistema de E/S en Perl 5.8.0.
- La E/S funciona igual que antes, pero incorpora características nuevas que permiten tratar la E/S como un conjunto de capas.
- Una capa de E/S, además de mover los datos, permite transformarlos: comprimir, descomprimir, cifrar, descifrar, convertir de una codificación a otra...
- Al leer un archivo que tiene una codificación específica, los datos no se convierten automáticamente en datos Unicode. Hay que especificar la capa apropiada al abrir los archivos.
- Los datos leídos se convierten a Unicode. Éste es el paso más importante para trabajar con Unicode.
- Estrategia recomendada: convertir en la entrada y en la salida.

Entrada/Salida

- Abrir para lectura un archivo codificado con UTF-8:
`open my $fh, '<:utf8', $input;`
- Abrir para lectura un archivo con otra codificación:
`open my $fh, '<:encoding(EUC-JP)', $input;`
- Abrir para escritura un archivo con codificación Latin-1:
`open my $fh, '>:encoding(Latin-1)', $input;`
- Salvo en el caso de la capa **:utf8**, que siempre se debe especificar así, hay flexibilidad para escribir los nombres de las codificaciones (no se distinguen mayúsculas de minúsculas y se pueden utilizar los distintos alias de una codificación).

Entrada/Salida

- Especificar UTF-8 como capa predeterminada de E/S:

```
use open ':utf8';
```

- Para escribir automáticamente datos UTF-8 en los identificadores de archivo estándar, la capa `open()` predeterminada y `@ARGV`, se puede utilizar el modificador de línea de comandos `-C` o la variable de entorno `PERL_UNICODE`.

Entrada/Salida

- Al escribir cadenas Unicode sin utilizar una capa PerlIO, se escribirán los bytes utilizados internamente (tanto del juego de caracteres nativo como UTF-8) y se mostrará una advertencia “Wide character” para los caracteres con código mayor que 0x0FF. Para evitarlo:

```
binmode STDOUT, ":utf8";
```

- **binmode** permite cambiar la codificación de una secuencia de datos (stream) abierta.
- Para adaptar código que funciona con la versión 5.6:

```
if ($] > 5.007) {  
    binmode $fh, ":utf8";  
}
```

Entrada/Salida - Ejemplo

Nota: utilizo código de una discusión de PerlMonks.

Queremos tokenizar cadenas de datos codificadas con ISO 8859-1 de un archivo de texto. Por simplificar, tokenizamos una sola cadena que contiene caracteres extendidos:

“Estoy realizando experimentos de codificación para la charla de mañana.”

Entrada/Salida - Ejemplo

Si lo hacemos como siempre:

```
#!/usr/bin/perl
use strict;
open IN, "<", $ARGV[0];
while (<IN>) {
    my @words = ( $_ =~ /\b(\w+)\b/g );
    print join "\n", @words;
    print "\n";
}
```

Entrada/Salida - Ejemplo

Resultado:

Estoy
realizando
experimentos
de
codificaci
n
para
la
charla
de
ma
ana

Entrada/Salida - Ejemplo

El resultado que buscamos:

Estoy
realizando
experimentos
de
codificación
para
la
charla
de
mañana

Lo podemos obtener de varias maneras:

Entrada/Salida - Ejemplo

Mediante el módulo Encode:

```
#!/usr/bin/perl
use strict;
use Encode;
open IN, "<", $ARGV[0];
while (<IN>) {
    my $utf8 = decode( 'iso8859-1', $_ );
    my @words = ( $utf8 =~ /\b(\w+)\b/g );
    print join "\n", map { encode( 'iso8859-1', $_ ) } @words;
    print "\n";
}
```

Entrada/Salida - Ejemplo

Mediante Perlio:

```
#!/usr/bin/perl
use strict;
open IN, "<:encoding(iso8859-1)", $ARGV[0];
binmode STDOUT, ":encoding(iso8859-1)";
while (<IN>) {
    my @words = ( /\b(\w+)\b/g );
    print join "\n", @words;
    print "\n";
}
```

Generar caracteres Unicode

- Para crear caracteres Unicode con código de carácter mayor que 0xFF (255 decimal) en literales de cadena, se interpolan los códigos de los caracteres con la notación `\x{hexcode}`
- `\x..` (sin `{}` y con sólo dos dígitos hexadecimales), `\x{}` y `chr()` devuelven para argumentos inferiores a 0x100 (256 decimal) un carácter de 8 bits, por compatibilidad con versiones anteriores de Perl. Para valores mayores, devuelven caracteres Unicode.

Generar caracteres Unicode

- Utilizando los nombres en lugar de los códigos:
use charnames ':full';
my \$arabic_alef = "\N{ARABIC LETTER ALEF}";
- Si el nombre Unicode es de la forma *< sistema de escritura >* *< nombre letra >* o *< sistema de escritura >* *< nombre de letra mayúscula/minúscula >*, se puede utilizar una forma abreviada:
use charnames ':short';
binmode STDOUT, ":utf8";
print "\N{greek:omega}\n";

Generar caracteres Unicode

- O bien:

```
use charnames qw(greek);  
binmode STDOUT, ":utf8";  
print "\N{omega}\n";
```

- Si hubiéramos escrito `\N{Omega}`, hubiera generado una omega mayúscula.
- Esta notación se puede utilizar también en las expresiones regulares.

Generar caracteres Unicode

- El pragma `charnames` también ofrece las funciones **`viacode`** y **`vianame`**:

`charnames::viacode(código)`

Devuelve el nombre correspondiente al código.

`my $code = charnames::vianame(nombre)`

Devuelve el código correspondiente al nombre.

Funciones

- **chr**(*hexcode*) devuelve el carácter correspondiente al código de carácter.
- **ord**(*char*) devuelve el código de carácter correspondiente al carácter.
- **index**, **length** y **substr** se aplican a caracteres Unicode, no a bytes.
- Otras funciones que cambian a semántica de caracteres al recibir datos Unicode: **chop**, **chomp**, **substr**, **pos**, **index**, **rindex**, **sprintf**, **write**.
- Las funciones que trabajan a nivel de bits, como **sort**, no cambian de semántica. En este caso, el equivalente Unicode lo proporciona el módulo `Unicode::Collate`.

Funciones

- Perl considera a los componentes de un carácter compuesto como caracteres independientes:

```
use charnames ':full';  
my $char = "\N{LATIN CAPITAL LETTER A}" .  
          "\N{COMBINING ACUTE ACCENT}";  
print length($char), "\n";
```

El resultado es 2, no 1.

- **Equivalencia** (eq, ne) y **ordenación** (lt, le, cmp, ge, gt) de cadenas: la comparación se basa en los códigos de los caracteres. Hay que tener en cuenta la normalización.
- Conversión de **mayúsculas y minúsculas**: las funciones como uc() o \U utilizan las tablas de conversión de mayúsculas y minúsculas de Unicode.

Expresiones regulares

- Funcionan igual con caracteres Unicode (p. ej., se puede utilizar `\w` para detectar un ideograma japonés).
- Si los datos son Unicode, se cambia automáticamente a modo de caracteres.
- `\X` permite detectar caracteres compuestos (un carácter base + caracteres de acentuación). Se debe utilizar en lugar del metacarácter “.”
- `\C` fuerza la detección de un solo byte.

Regexp: propiedades de caracteres Unicode

- Las clases de caracteres (`/[a-z]/`) y `tr/ / /` se aplican a caracteres en lugar de a bytes. Para especificar clases de caracteres en expresiones regulares es conveniente utilizar las **propiedades** de caracteres Unicode.
- Ejemplos de propiedades:

Nombre corto	Nombre largo
L	Letter
Lu	UppercaseLetter
M	Mark (signo de acentuación)
P	Punctuation
N	Number
AN	Arabic Number

Regexp: propiedades de caracteres Unicode

- También incluyen nombres de bloques y de sistemas de escritura.
- Los nombres de bloque tienen el prefijo "In":
InHiragana, InKatakana, InArabic, InBasicLatin, InCurrencySymbols, InTags
- Para detectar una propiedad: `\p{}` (si el nombre de propiedad tiene una sola letra, se pueden omitir las llaves).
- Para detectar la negación de una propiedad: `\P{}`.
También se puede utilizar `^` dentro de `\p{}` o `\P{}` para negar. `\p{^Cyrillic}` equivale a `\P{Cyrillic}`.
- Es posible crear propiedades definidas por el usuario.

Problemas

- Si se utilizan locales, Perl será más lento al procesar datos Unicode.
- Algunas funciones (p. ej., length, substr, index o las expresiones regulares) son más lentas al trabajar con cadenas codificadas en UTF-8 en lugar de bytes.
- Las propiedades Unicode (equivalentes a clases de caracteres) son más lentas, pero normalmente estas propiedades abarcan muchos más caracteres que el equivalente no Unicode. Ejemplo: Nd representa a 268 caracteres, mientras que d representa a 10 caracteres ASCII.

Problemas

- Al intercambiar datos con extensiones (módulos que utilizan código XS/C) que no están preparadas para trabajar con el indicador (flag) UTF-8, los datos devueltos podrían tener desactivado dicho indicador.
- Ejemplos: HTML::Parser, DBI, Text::CSV_XS
- Solución: activar explícitamente el indicador en los datos devueltos:

```
my $string = Encode::decode_utf8( $input );
```

o bien

```
Encode::_utf8_on( $input );
```

Módulos de CPAN

- **Encode** - Character encodings
- **Encode::Unicode** - Various Unicode Transformation Formats
- **Unicode::UCD** - Unicode character database
- **Unicode::Transform** - Conversion among Unicode Transformation Formats
- **Unicode::Normalize** - Unicode Normalization Forms
- **Unicode::Decompose** - Unicode decomposition and normalization
- **Unicode::Collate** - Unicode Collation Algorithm

Módulos de CPAN

- **Unicode::CharName** - Look up Unicode character names
- **Unicode::Char** - OO interface to charnames and others
- **Unicode::MapUTF8** - Conversions to and from arbitrary character sets and UTF8
- **Unicode::Map8** - Mapping table between 8-bit chars and Unicode
- **Unicode::String** - String of Unicode characters (UTF-16BE)
- **Convert::Scalar** - Convert between different representations of Perl scalars

Referencias

- perluniintro, perlunicode, Encode::PerlIO
- www.unicode.org
- "Advanced Perl Programming" Simon Cozens, O'Reilly
- "Unicode in Perl" Simon Cozens, The Perl Journal (September 2004)
- "Pro Perl" Peter Wainwright, Apress
- <http://ahinea.com/en/tech/perl-unicode-struggle.html>
- www.perlmonks.org