

# Unicode en Perl

Enrique Nell

Barcelona Perl Mongers

Octubre de 2008

# Implementación en Perl

- Para trabajar con Unicode en Perl hay que utilizar como mínimo la versión 5.8.2.
- La carpeta unicore (v5.8/ v5.10) [o unicode (v5.6)] de la biblioteca estándar de Perl contiene una representación del estándar Unicode en varios archivos de texto manipulables por un programa.

En Perl 5.10 se ha actualizado a la versión 5.0.0 del estándar.

- Se puede utilizar el módulo `Unicode::UCD` para consultar esta información.

# Pragma utf8

- En la versión 5.6 servía para declarar que las operaciones de un bloque o archivo utilizaban Unicode.
- A partir de la versión 5.8, esto lo indican los mismos datos => ahora este pragma sirve para poder utilizar caracteres UTF-8 en identificadores, literales de cadena y expresiones regulares. Ejemplo:

```
use utf8;
```

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

```
print "ネルエンリケは翻訳家です\n";
```

# Bytes o caracteres

- Al procesar texto, Perl permite trabajar con bytes o con cadenas de caracteres Unicode.
- En futuras versiones sólo se utilizarán caracteres Unicode.
- Si puede determinar que los datos de entrada son caracteres Unicode, cambia a semántica de caracteres. Si no, utiliza semántica de bytes.

# Bytes o caracteres (cont.)

- Internamente, Perl utiliza el juego de caracteres de 8 bits nativo de la plataforma y también utiliza UTF-8 para codificar caracteres Unicode.

Cuando Perl lee una cadena, trabajará mientras pueda con la codificación de 8 bits (normalmente ISO-8859-1, pero se puede utilizar otra codificación con el pragma encoding).

Cuando deja de ser posible (p. ej., al añadir caracteres no-Latin-1 a la cadena), convierte el formato interno de la cadena a UTF-8.

# Bytes o caracteres (cont.)

- En Perl, las cadenas de texto Unicode tienen un indicador (flag) que indica que la representación interna de la cadena es UTF-8.

Para que Perl trate los caracteres de una cadena UTF-8 como caracteres Unicode, la cadena debe tener activado este indicador. Si no, usa ISO-8859-1 (semántica de bytes).

- 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`)

# Bytes o caracteres (cont.)

- La semántica utilizada en operaciones con cadenas de caracteres Unicode depende de la codificación interna de las cadenas.
- En caso de duda, conviene utilizar la función `up` del módulo `Unicode::Semantics` o la función `utf8::upgrade` para convertir el formato interno de la cadena a UTF-8.

Si la cadena ya está marcada como UTF-8, no se vuelve a convertir.

# Bytes o caracteres (cont.)

- **Problema:** 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.

**Solución:** activar explícitamente el indicador en los datos devueltos:

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

o bien

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

# Entrada/Salida

- La estrategia recomendada para operaciones que requieran E/S de texto en un programa es convertir en la entrada y en la salida:

1. Leer y descodificar ( binario => texto )

2. Procesar ( texto )

3. Codificar y escribir ( texto => binario )

**bytes -> caracteres -> bytes**

# Entrada/Salida (cont.)

- PerlIO: nuevo sistema de E/S incorporado 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...

# Entrada/Salida (cont.)

- Al leer un archivo con una codificación específica, los datos no se convierten automáticamente en texto Unicode. Hay que especificar la capa apropiada:

Abrir para lectura un archivo UTF-8:

```
open my $fh, '<:encoding(UTF-8), $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;
```

# Entrada/Salida (cont.)

- 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 (p. ej., Latin-1 o iso88591-1).

# Entrada/Salida - Ejemplo

- Queremos tokenizar cadenas de datos codificadas con ISO-8859-1 de un archivo de texto.

Por simplificar, utilizamos un archivo que contiene una sola cadena con caracteres extendidos:

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

# Entrada/Salida - Ejemplo

- Si lo hacemos como siempre (semántica de bytes):

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

# Entrada/Salida - Ejemplo

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

\w no reconoce los caracteres extendidos (ó, ñ) como caracteres alfabéticos. Tenemos que usar semántica de caracteres.

# Entrada/Salida - Ejemplo

Podemos obtener el resultado correcto con el módulo Encode:

```
use Encode;
```

```
open my $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";
```

```
}
```

```
close $in;
```

# Entrada/Salida - Ejemplo

O con PerlIO:

```
open my $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";
}
close $in;
```

# Entrada/Salida (cont.)

- Se puede 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 (cont.)

- **Problema:** 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 0xFF.

**Solución:** utilizar `binmode STDOUT, ":utf8";`

`binmode` permite cambiar la codificación de una secuencia de datos (stream) abierta.

# 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 un carácter de 8 bits para argumentos inferiores a 0x100 (256 decimal), por compatibilidad con versiones anteriores de Perl.

Para valores mayores, devuelven caracteres Unicode.

# Generar caracteres Unicode (cont.)

- Se pueden utilizar los **nombres** de los caracteres en lugar de los códigos:

```
use charnames ':full';
```

```
my $arabic_alef = "\N{ARABIC LETTER ALEF}";
```

# Generar caracteres Unicode (cont.)

- Si el nombre Unicode es de la forma  
<sisistema de escritura> <nombre letra> o  
<sisistema de escritura> <nombre 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.

# Expresiones regulares

- Si los datos son Unicode, se cambia automáticamente a modo de caracteres.
- Las expresiones regulares funcionan igual con caracteres Unicode (p. ej., se puede utilizar `\w` para detectar un ideograma japonés).

Las clases de caracteres (`[a-z]`) y `tr / / /` se aplican a caracteres, no a bytes.

# Expresiones regulares (cont.)

- \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.

# Propiedades de caracteres

- Para especificar clases de caracteres en expresiones regulares conviene utilizar **propiedades** de caracteres Unicode.

Ejemplos:

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

# Propiedades de caracteres

- También incluyen nombres de bloques y de sistemas de escritura.
- Los nombres de bloque tienen el prefijo "In": InHiragana, InKatakana, InArabic, InBasicLatin, InCurrencySymbols, InTags

# Propiedades de caracteres (cont.)

- 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.

# Propiedades de caracteres (cont.)

- El módulo `Unicode::Properties` permite obtener las propiedades de un carácter determinado:

```
use 5.010;
```

```
use utf8;
```

```
use Unicode::Properties 'uniprops';
```

```
my @props = uniprops('ホ');
```

```
say join ", ", @props;
```

```
# Resultado: Alphabetic, Any, Assigned, IDContinue, IDStart,  
InKatakana, Katakana
```

# Propiedades de caracteres (cont.)

- Las operaciones con propiedades Unicode (equivalentes a clases de caracteres) son más lentas, pero normalmente estas propiedades abarcan muchos más caracteres que el equivalente no Unicode.

Ejemplo: `\p{Nd}` representa a 268 caracteres, mientras que `\d` sólo representa a 10 caracteres ASCII.

# 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.

# Funciones

- 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.

# Módulos de CPAN

- **Encode** - Character encodings
- **Encode::Unicode** - Various Unicode Transformation Formats
- **Unicode::UCD** - Unicode character database
- **Unicode::Semantics** - Work around \*the\* Perl 5 Unicode bug
- **Unicode::Properties** - find out what properties a character has

# 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)

# Módulos de CPAN

- **Convert::Scalar** - Convert between different representations of Perl scalars
- **Unicode::Transform** - Conversion among Unicode Transformation Formats
- **Unicode::Normalize** - Unicode Normalization Forms
- **Unicode::Decompose** - Unicode decomposition and normalization
- **Unicode::Collate** - Unicode Collation Algorithm

# Referencias

- perlunitut, perlunifaq, perluniintro, perlunicode, Encode::PerlIO
- [www.unicode.org](http://www.unicode.org)
- "Advanced Perl Programming" Simon Cozens, O'Reilly
- Juerd Waalboer's Perl Unicode Advice page:  
<http://juerd.nl/site.plp/perluniadvice>

# Referencias

- "Pro Perl" Peter Wainwright, Apress
- "Unicode in Perl" Simon Cozens, The Perl Journal (September 2004)
- <http://ahinea.com/en/tech/perl-unicode-struggle.html>
- [www.perlmonks.org](http://www.perlmonks.org)