Cheryl lee - www.sxc.hu

Organização de conjuntos de caracteres com Perl

Sopa de letrinhas

Quando caracteres estranhos aparecem no código ou nos dados de um programa, os programadores Perl precisam de uma solução que evite as complicações do Babel.

por Michael Schilli

No princípio havia a tabela ASCII – 128 caracteres que permitiam a escrita de textos em língua inglesa, incluindo alguns caracteres presentes em qualquer máquina de escrever, como % e $, e também alguns caracteres de controle como quebras de linha e página, ou ainda um apito. Era apenas uma questão de tempo até que falantes de outras línguas desejassem escrever caracteres acentuados, cedilhas, tremas etc. A primeira forma de tratar esse problema foi agrupar a todos no próximo conjunto de 128 caracteres. Todos esses 256 caracteres foram numerados de 0 a 255 e codificados em computadores com 8 bits (1 byte) de dados.

Exemplo 1: latin1

```
01 #!/usr/bin/perl -w
02 use strict;
03 my $s = "ü";
04 print "utrema=$s", "\n";
```

Daí surgiu o padrão *ISO 8859*, também conhecido como *Latin 1*.

Línguas diferentes

Tudo começou com o *ISO-8859-1*, mas ao longo do tempo vieram outras variantes, chegando até o valor atual de *ISO-8859-15*, que ainda inclui o caractere do euro. Entretanto, a maioria dos navegadores web atuais não utiliza o padrão ISO-8859-1 para decodificar os conteúdos que adotam a codificação ISO-8859-1. Em vez disso, eles empregam o padrão *Windows-1252*, que inclui alguns caracteres adicionais, como o euro. Obviamente o resto do mundo não quis ficar para trás, e a corrida para exibir conjuntos de caracteres cada vez mais complexos começou.

Esquemas de codificação para línguas asiáticas, como *Shift-JIS* e *BIG5*, por exemplo, foram criados. Porém, os desenvolvedores logo perceberam que isso não os estava levando a lugar algum, e assim criaram o *Unicode*, uma enorme tabela que contém todos os caracteres das línguas comuns do mundo.

UTF 8

O padrão *UTF 8* oferece uma abordagem para a codificação da tabela Unicode em computador. Se tivesse surgido uma súbita ne-

```
mschilli@seraph:~/DEV
iso-8859-1 $ perl -le 'print chr(0xfc)'
ü
iso-8859-1 $ perl -le 'print chr(0xc3), chr(0xbc)'
Ã¼
iso-8859-1 $
```

Figura 1 Em um terminal configurado com *ISO-8859-1*, a saída em *Latin1* fica boa, mas a saída em *UTF 8* não faz sentido.

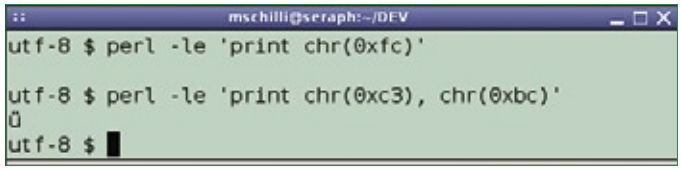

Figura 2 Em um terminal configurado com *UTF 8*, a saída em UTF 8 fica boa, mas os caracteres codificados em *ISO-8859-1* não são mostrados.

cessidade de se codificar caracteres ASCII em dois ou quatro bytes, os requisitos de memória teriam aumentado vertiginosamente. Para manter a possibilidade de exibir a antiga tabela ASCII com um único byte, a tabela UTF 8 foi projetada para que os primeiros 128 caracteres fossem os mesmos presentes na tabela ASCII.

Todavia, o próximo grupo de 128 caracteres é composto por códigos especiais de mascaramento que indicam que um número específico de códigos adicionais se segue, de forma a identificar inequivocamente qual caractere da tabela Unicode deve ser exibido. Por exemplo, o *ç* é armazenado sob o código 231 (*oxE7*) na tabela ISO-8859-1. Se você possuir um texto ISO-8859-1 que contenha um byte com o valor de *oxE7*, o caractere obviamente é um *ç*.

Um A com til

Na codificação UTF 8, o *ü* é representado por dois bytes – 195 e 198 (*oxC3* e *oxBC*). Se você possui um texto UTF 8 e o computador primeiro vê um byte com o valor de *oxC3* seguido por um de *oxBC*, ambos serão igualmente mostrados como a letra *ü*.

Por outro lado, se a codificação não estiver clara e o computador vir um byte *oxC3*, a questão será se isso é o primeiro byte de um *u* com trema em formato UTF 8 ou um byte ISO-8859-1 que representa um caractere completo. Se o byte inicial *oxC3* de uma seqüência UTF 8 for interpretado incorretamente como ISO-8859-15, será exibido o caractere que representa o terror dos programadores que brigam com as diferentes codificações – o Ã.

Embora esse caractere faça parte de nosso alfabeto, ele não fará sentido se for exibido incorretamente no lugar de um *ü*, *ç* ou qualquer outro. Quando ele é visto nessas condições, isso significa que provavelmente o texto está codificado em UTF 8, mas está sendo interpretado como ISO-8859-1.

Terminal

Se o objetivo for mostrar no terminal a saída de texto de um programa, então o terminal precisa saber como interpretar a cadeia de bytes gerada pelo programa, de forma a localizar os caracteres corretos para exibir.

Para iniciar um terminal X com suporte a UTF 8, pode-se rodar `xterm -u8 +lc`. A primeira opção ativa o suporte a UTF 8, e a segunda desabilita a interpretação de variáveis de ambiente, como `LANG`, para evitar que elas interfiram nesse processo.

Para abrir um terminal em modo ISO-8859-1, execute o comando `xterm` com a variável `LANG` definida, como por exemplo:

```
LANG=en_US.ISO-8859-1 xterm
```

As **figuras 1** e **2** mostram a saída de dois scripts *Perl* em um terminal ISO e outro UTF 8, respectivamente. Um único byte com o valor de *oXE7* é corretamente interpretado como *ç* pelo terminal ISO vermelho. Entretanto, a seqüência *oxC3BC* é exibida como um Ã e a representação ISO de *oXBC*, o caractere para 1/4.

Já o terminal UTF 98 verde não exibe byte algum com o valor de *oxFC*, mas a seqüência *oxC3BC* exibe um *ü* como esperado.

Latim como padrão

A menos que o Perl receba ordens para se comportar de outra forma, ele interpretará o código-fonte de um script, incluindo todos os seus caracteres, expressões regulares, variáveis e nomes de funções com o código ISO-8859-1.

Se for usado um editor configurado para essa codificação, o *ü* no texto do programa será exibido com o código *oxFC*, como mostra o utilitário *hexdump* (**figura 3**).

Porém, o **exemplo 2** foi criado no editor *vim* com o comando `set encoding=utf-8`. As marcações vermelhas na saída do hexdump na **figura 3** mostram que o trema no texto do código-fonte do programa foi codificado em dois bytes: *C3* e *BC*.

A **linha 3** do **exemplo 2** é responsável por fazer o Perl interpretar o código-fonte como *UTF 8* (`use utf8`). Isso garante que o texto `ü`, representado por *oxC3BC* no código, exiba um único caractere – o equivalente Unicode a *ü*. Como resultado, `length($s)` não retornará um valor de 2, mas apenas 1.

A seguir, a **linha 6** especifica a forma de saída padrão para UTF 8 (`binmode(STDOUT, ":utf8")`). Isso assegura que o Perl mostrará saídas Unicode no formato UTF 8

Exemplo 2: utf8

```
01 #!/usr/bin/perl -w
02 use strict;
03 use utf8;
04 my $s = "ü";
05
06 binmode STDOUT, ":utf8";
07 print "utrema=$s", "\n";
```

Exemplo 3: espiar

```
01 #!/usr/bin/perl -w
02 use strict;
03 use utf8;
04 use Data::Hexdumper;
05 use Encode qw(_utf8_off is_utf8);
06
07 my $s = "ü";
08
09 if( is_utf8($s) ) {
10   print "sinal UTF-8 estah 'ligado'.\n";
11 }
12
13 print "Comprimento: ", length($s), "\n";
14 _utf8_off($s);
15 print "Comprimento: ", length($s), "\n";
16
17 print hexdump(data => $s), "\n";
```

Exemplo 4: isotest.cgi

```
01 #!/usr/bin/perl -w
02 use strict;
03 use CGI qw(:all);
04
05 print header(
06   -type => 'text/html',
07   -charset => 'iso-8859-1');
08
09 print "O simbolo do Euro eh ",
10   chr(0x80), ".\n";
```

e, portanto, que o terminal receberá os dados UTF 8 que espera, exibindo-os corretamente. Sem a chamada de `binmode`, o Perl tentaria converter o texto de saída para Latin 1, o que faria sentido em terminais ISO-8859-1, mas não em um UTF 8.

E tudo começa a piorar se o caractere Unicode não puder ser convertido para Latin 1, como o *katakana* japonês "me" que possui um número Unicode de *30E1*. Nesse caso, pode-se ver um aviso *Wide character in print*. O uso de `binmode(STDOUT)` para especificar a disciplina de linha para o arquivo de saída impede que o Perl tente converter a saída, fazendo com que ele mostre um texto UTF 8 cru. Em terminais UTF 8, essa é exatamente a estratégia a ser usada.

Referência

A página de manual `man iso-8859-1` detalha a codificação Latin 1. Referindo-se ao número octal 347, por exemplo, ou ao valor hexadecimal *E7*, pode-se ver uma entrada para *LATIN SMALL LETTER C WITH CEDILLA*, ou um *ç*.

Se for necessário acessar a tabela Unicode, o arquivo `unicore/UnicodeData.txt`, normalmente em `/usr/lib/perl5/5.8.x/`, é o local certo para procurar. Novamente, há uma entrada para o número *00E7: LATIN SMALL LETTER C WITH CEDILLA*.

A **figura 4** mostra que, logo abaixo do número para o *ü* minúsculo na tabela Unicode está o número para o mesmo caractere na tabela ISO-8859-1, que é o mesmo nas duas tabelas, pois os criadores do Unicode modelaram os primeiros 256 bytes de acordo com o padrão ISO-8859-1.

Note que o número Unicode não representa a codificação Unicode do caractere. Por exemplo, *ü*, o caractere Unicode com o número *00FC*, é representado como *C3BC* em UTF 8. Como mencionado antes, o UTF 8 é apenas uma forma mais eficaz de se codificar a tabela Unicode.

Entrada e saída

Quando um programa em Perl lê ou escreve dados, o programador precisa especificar o formato de entrada ou saída para os dados. Para ler as linhas de um arquivo codificado em UTF 8, pode-se usar o truque do `binmode` do **exemplo 2** ou especificar a disciplina de linha usando um comando `open` com três parâmetros:

```
open ARQUIVO, "<:utf8", "arquivo.
➥ txt"
```

Se o programa depois ler uma linha do arquivo com `<ARQUIVO>` e atribuir o resultado a um escalar, pode-se ter certeza de que a cadeia de caracteres é Unicode, e o Perl tomará nota desse fato internamente.

```
mschilli@seraph:~/DEV
$ cmdcolor code.term
$ hexdump -C latin1
00000000  23 21 2f 75 73 72 2f 62  69 6e 2f 70 65 72 6c 20  |#!/usr/bin/perl |
00000010  2d 77 0a 75 73 65 20 73  74 72 69 63 74 3b 0a 6d  |-w.use strict;.m|
00000020  79 20 24 73 20 3d 20 22  c3 bc 22 3b 0a 70 72 69  |y $s = "..";.pri|
00000030  6e 74 20 22 75 74 72 65  6d 61 3d 24 73 22 2c 20  |nt "utrema=$s", |
00000040  22 5c 6e 22 3b 0a                                 |"\n";.|
00000046
$ hexdump -C utf8
00000000  23 21 2f 75 73 72 2f 62  69 6e 2f 70 65 72 6c 20  |#!/usr/bin/perl |
00000010  2d 77 0a 75 73 65 20 73  74 72 69 63 74 3b 0a 75  |-w.use strict;.u|
00000020  73 65 20 75 74 66 38 3b  0a 6d 79 20 24 73 20 3d  |se utf8;.my $s =|
00000030  20 22 c3 bc 22 3b 0a 0a  62 69 6e 6d 6f 64 65 20  | "..";..binmode |
00000040  53 54 44 4f 55 54 2c 20  22 3a 75 74 66 38 22 3b  |STDOUT, ":utf8";|
00000050  0a 70 72 69 6e 74 20 22  75 74 72 65 6d 61 3d 24  |.print "utrema=$|
00000060  73 22 2c 20 22 5c 6e 22  3b 0a                    |s", "\n";.|
0000006a
$
```

Figura 3 Um script *Perl* escrito em *ISO-8859-1* usará o código *0xFC* para representar o *ü* do código-fonte. Se o código-fonte for escrito em *UTF 8*, o *ü* é representado por uma seqüência dos bytes *0xC3BC*.

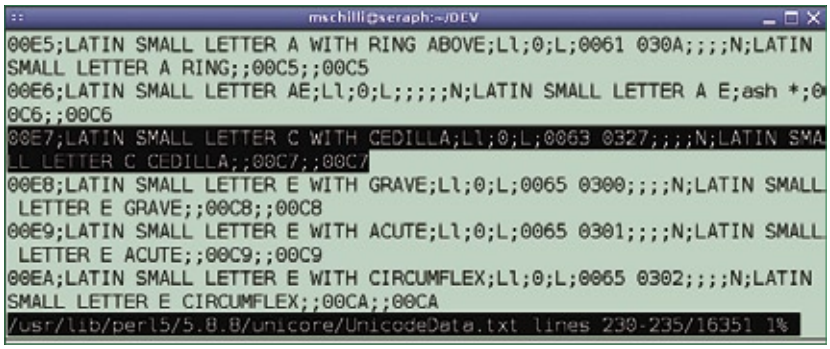

Figura 4 A entrada do caractere *ü* na tabela *Unicode* do *Perl*.

Sem a disciplina de linha, a entrada seria interpretada como ISO-8859-1, e o Perl colocaria os bytes crus em um escalar de cadeia de caracteres sem marcá-lo como UTF 8. O mesmo princípio se aplica à saída. Um `>:utf8` ou `> >:utf8` no segundo parâmetro do `open` especifica a disciplina de linha da saída para o modo UTF 8, e um `print FILE $string` exibirá como saída a cadeia de caracteres formatada em UTF 8 sem alterações. Como alternativa, pode-se usar `binmode` para modificar o manipulador do arquivo.

O último véu

O **exemplo 3** mostra como o Perl lida internamente com cadeias de caracteres Unicode. Devido ao pragma `use utf8` especificado antes, o caractere *ü* (digitado em UTF 8 no *vim*) é identificado e gerenciado como uma cadeia de caracteres Unicode.

Para permitir que isso ocorresse, o Perl atribuiu um sinal interno que pode ser consultado (`is_utf8()`) e manipulado (`_utf8_off()`) com o módulo *Encode*.

A saída de *espiar* na **figura 5** mostra que a cadeia UTF 8 realmente possui comprimento de 1. Se o sinal for apagado com `_utf8_off()`, o comprimento da cadeia de repente cresce para dois caracteres.

A saída do módulo do CPAN *Data::Hexdumper* informa que a cadeia agora está armazenada internamente como *oxC3BC* – e que realmente é UTF 8.

O **exemplo 4** mostra como um script *CGI* promete entregar ao navegador texto codificado em ISO-8859-1, mas depois envia um caractere de euro com o código de *ox80*, o que está de acordo com o padrão Windows-1252[1]. Como se pode ver na **figura 6**, o navegador generosamente concorda em mostrar o caractere do euro.

Se o script do servidor especificasse ISO-8859-1 em seu cabeçalho, veríamos um ponto de interrogação preto no lugar do símbolo do euro. Esse símbolo possui o código *oxA4* na tabela ISO-8859-1. Se o código for modificado para refletir isso, o navegador novamente mostrará o símbolo do euro corretamente.

Não tão generoso

A biblioteca cliente web de Perl, *LWP*, não é tão generosa. O **exemplo 5** mostra a saída como UTF 8 e até especifica o cabeçalho corretamente.

O caractere de euro na cadeia é representado por seu número de série Unicode *\x{20AC}*. Entretanto, há diversos aspectos a observar no cliente da aplicação web, se o texto da página web for codificado como

Exemplo 5: isotest2.cgi

```
01 #!/usr/bin/perl -w
02 use strict;
03 use CGI qw(:all);
04
05 print header(
06   -type => 'text/html',
07   -charset => 'utf-8');
08
09 binmode STDOUT, ":utf8";
10 print "O simbolo do Euro eh ",
11   "\x{20AC}.\n";
```

Exemplo 6: clienteweb

```
01 #!/usr/bin/perl -w
02 use strict;
03 use LWP::UserAgent;
04
05 my $ua = LWP::UserAgent->new(
06   parse_head => 0
07 );
08
09 my $resp = $ua->get(
10   "http://perlmeister.com/cgi/isotest.cgi");
11
12 if($resp->is_success()) {
13   my $text = $resp->decoded_content();
14   binmode STDOUT, ":utf8";
15   print "$text\n";
16 }
```

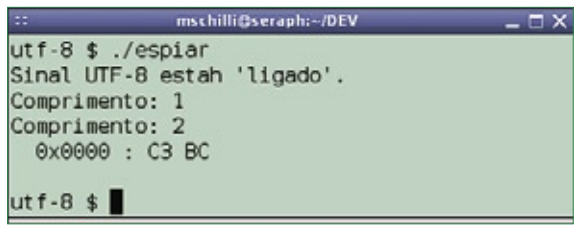

Figura 5 Em uma cadeia *Unicode*, um caractere multi-byte realmente possui comprimento 1. Se removermos a propriedade Unicode da cadeia, o *Perl* interpretará os bytes individualmente.

Figura 6 O navegador mostra o símbolo do euro.

UTF 8 no lado do servidor. A idéia é usar a biblioteca LWP para acessar a página a partir do servidor web e, se tudo funcionar direito, para armazenar uma cadeia Unicode em Perl. O **exemplo 6** demonstra a técnica.

Devido a uma falha conhecida na biblioteca LWP (ou em `HTML::HeadParser`, para ser mais preciso), o Perl exibe um horrendo aviso *"Parsing of undecoded UTF-8 will give garbage when decoding entities"* ao retornar UTF 8. Isso pode ser evitado especificando-se a opção `parse_head => 0` na chamada do construtor do `UserAgent`[2].

Para permitir que o Perl guarde o texto UTF 8 retornado em uma cadeia Unicode, é necessário evitar usar o típico método `content()` para extrair o texto da página a partir do objeto `HTTP::Response`. O melhor é utilizar `decoded_content()` em seu lugar.

Esse método usa o campo `charset` da resposta do servidor web para adivinhar como decodificar seu conteúdo. Enquanto o cliente continuar honrando a disciplina de linha para `STDOUT`, não há nada para impedir a exibição correta em um terminal configurado para o modo UTF 8.

Conclusões

Caminhar pelos mundos da codificação sempre foi um problema. Mas se você preferir não restringir a disponibilidade de seu software para uma fração do mercado, é bom adotar seriamente uma estratégia de internacionalização. ■

Mais Informações

[1] Windows-1252: http://en.wikipedia.org/wiki/Windows-1252

[2] Falha conhecida na LWP: http://www.mail-archive.com/libwww@perl.org/msg06330.html

O autor

Michael Schilli trabalha como desenvolvedor de software para o *Yahoo!*, em Sunnyvale, Estados Unidos. É autor de "Perl Power", publicado pela editora Addison-Wesley. Sua homepage é http://perlmeister.com.