

Unicode in Perl Regexes

ἰβυοαμ λλqἰπποῦ sεοb
 zuoἰssεαpxε πλνbελ
 7noqε mεηκ noλ 7ηbnoῦ7
 noλ βυἰη7λπελ εηηM

Tom Christiansen

<tchrist@perl.com>

Setup

- Pragmatically, you want at least use v5.14, use utf8, and

```
use charnames qw(:full);
```

- Always decode incoming data and encode outgoing data.

```
use open qw< :encoding(UTF-8) :std >;
```

```
$ export PERL_UNICODE="AS"
```

- Normalize incoming data to NFD, outgoing data to NFC.

```
while (<>) {
    chomp;
    $_ = NFD($_); # from Unicode::Normalize
    ...
} continue {
    say NFC($_);
}
```

“Characters”: Graphemes or Code Points?

- In Unicode, a single user-visible character — a *grapheme* — like \tilde{o} may (or may not!) occupy multiple code points. That grapheme there takes anywhere from 1–3 code points, depending on normalization: "\x{22D}" in NFC, "\x{6F}\x{303}\x{304}" in NFD, or "\x{F5}\x{304}", which is neither.
- Use \X in a regex to match any of those. It also matches an entire "\r\n". (*Very* technically speaking, Perl’s \X matches what the Unicode Standard refers to as an *extended* grapheme cluster.)
- Don’t use \PM\pM*; that doesn’t really work out so well: you shouldn’t (pretend to) apply Marks to linebreaks!
- The Unicode::Normalize module has functions like NFD & NFC.

- Careful with its NFKC & NFKD functions, as those do discard information. Still, $\frac{3}{4} \Rightarrow 3/4$, $20^{th} \Rightarrow 20th$, & $hic sunt dracones \Rightarrow hic sunt dracones$ can be useful.

Dot à la Unicode

- Perl's dot `/./` to match "any one code point" never quite did that. Without `/s`, it means `[^\n]` — for which you can use a braceless `\N` these days. Not much call for `(-s:.)` though.
- Adding `/s` make dot now include linefeed, so `(?s:.)` means `\p{Any}`. With v5.14's use `re "/s"`, we may see more of it, but it's still (mostly) an ASCII-minded solution in a Unicode world.
- Like #1 but Unicode-savvy, `\V` matches any one code point *lacking* the `Vert_Space` property. It's the same as `\P{Vert_Space}`, or `[^\p{LB=CR}\p{LB=LF}\p{LB=NL}\p{LB=BK}]`.
- At last something reasonable: `\X` from the previous slide. **It's the first one that's safe on graphemes.** This is *usually* what you really want, not #2 or #3. Just depends, though.
- To modify #4 to omit `\R` per #1 & #3, use `(?!\R)\X` instead. Huffman loses by 12:1.

Line Endings

- Perl's `/^/` and `/$/` don't know about Unicode linebreaks.

```
/foo$/;      # old way
/foo\R?$/;   # for Unicode linebreaks
```

- Its `/\R/`, however, does. There are all the same:

```
\R
(?:\013\010|\v)
(?:\x0D\x0A|[\x0A-\x0D\x85\x{2028}\x{2029}])
```

- You could just convert them all.

```
s/\R/\n/g;    # convert Unicode linebreaks to \n
```

- Or, sometimes, perhaps like this.

```
@paras = split /\R+/, $file_contents;
```

Case in Unicode: Surprises

- Not all letters have case, like 文字化け and 東京.
- Not all things with case are letters:
 - `xii` & `XII` are lower- & uppercase numbers, not letters.
 - `Ⓚ` & `Ⓚ` are lower- & uppercase symbols, not letters.
 - `U+0345 COMBINING GREEK YPOGEGRAMMENI` is a lowercase combining mark, not a letter, yet whose uppercase *is* a letter, `GREEK CAPITAL LETTER IOTA`.
- Not all cased letters change case: `SMALL CAPS`, `supers`, and even `LETTER` are all completely lowercase strings — *and stay that way* even when you `uc` them!
- There are no roundtrip guarantees: you cannot depend on any of `$s`, `uc lc($s)`, and `lc uc($s)` being the same strings.
- String lengths may change: `length($s) != length(uc $s)`.

Three Cases in Unicode

- Unicode has *three* different cases, not just the two traditional ones:

	English	Spanish	German	Greek
lowercase	vis-à-vis	los ángeles	tschüß	ἄ στο διάολο
titlecase	Vis-À-Vis	Los Ángeles	Tschüß	Ἄ Στο Διάολο

uppercase	VIS-À-VIS	LOS ÁNGELES	TSCHÜSS	ΑΙ ΣΤΟ ΔΙΑΟΛΟ
-----------	-----------	-------------	---------	---------------

- Any of those also matches the other two in its column under the `/i` modifier. This is called *casefolding*, a regex concept.
- Case is *for the most part* unaffected by locale, so the language titles above are merely informative.
- Only the first letter in each word in the middle row is in titlecase.

Casefolding in Unicode

- There are two kinds of casefolding: *simple* casefolding, where string lengths won't change, and *full* casefolding, where they may.
- Perl supports full casefolding in its case-insensitive matches.
- These show that Perl understands not just simple casefolding:

```
"\N{KELVIN SIGN}" =~ /k/i
"pofT"           =~ /post/i
  ↪ (that's a LATIN SMALL LETTER LONG S)
```

- But that Perl also understands full casefolding:

```
"MASSE"          =~ /maße/i
"Ἄ ΣΤΟ ΔΙΑΟΛΟ"  =~ /ΑΙ ΣΤΟ ΔΙΑΟΛΟ/i
```

- See Unicode UAX #21 on Case Mappings for details.

Casemapping in Unicode

- When you convert a string to a different case, this is called *casemapping*.
 - The `lc` function, or the `\l` and `\L` interpolation escapes, map to lowercase.
 - The `ucfirst` function, or the `\u` interpolation escape, maps to titlecase.
 - The `uc` function, or the `\U` interpolation escape, maps to uppercase.

```
use v5.14;
use utf8;
use Unicode::Normalize;
for ( "mad dog", "Ἄ ΣΤΟ ΔΙΑΟΛΟ", "tschüß",
      scalar reverse(NFC "tschüß") )
{
    say s/(\w+)/\u\L$1/rg;
}
# Mad Dog, Ἄ ΣΤΟ ΔΙΑΟΛΟ, Tschüß, Ssühcst
```

- With `s/PATTERN/REPLACEMENT/i`, the *PATTERN* is subject to casefolding due to the `/i`, and the *REPLACEMENT* to casemapping if it uses `\L` &c.

Casing Concerns

- Partial matches of casefolded strings is not supported. Use casemapping instead:

```
"maße"  =~ /MASSE/i
"maße"  !~ /MAS/i
uc("maße") =~ /MAS/i
```

- Multicharacter casefolds are intentionally disabled in inverted character classes for reasons too complex to dwell upon here. See the *perldelta* manpage.
- Perl has no builtin support for conditional or locale-dependent casing rules:
 - Although Greek σ , ς , and Σ all casefold to the same thing in patterns, `lc` of Σ always casemaps to σ , never to ς . Thus the `lc` of $\Sigma\text{ΤΙΜΑΣ}$ shows up as *στιγμασ* instead of as *στιγμας* — let alone as *στιγματα*.
 - U+0130 LATIN CAPITAL LETTER I WITH DOT ABOVE as in *Istanbul* doesn't know its `lc` should in **Turkish languages alone** be just plain *istambul*, not `"i\N{COMBINING DOT ABOVE}stambul"`.
- See the CPAN `Unicode::Casing` module for those cases — and casings.

Character Classes: `|w|d|s|v|h`

- You probably figured out around two slides ago that `\w` was working correctly on Unicode. These things are all defined by Unicode UTS #18 Regular Expressions in its Annex C on Compatibility Properties.

http://unicode.org/reports/tr18/#Compatibility_Properties

- `\w` is anything that's an `Alphabetic` or a `Digit`, or has the General Category of `Mark` or of `Connector Punctuation`. There are *102,724 such code points* as of Unicode 6.0.0; `(?=\p{Latin})\w/` matches only 1,267 but misses `Digits`.
- `\d` is any of the 420 code points with the `Decimal Number` General Category.
- `\s` is *supposed* to be any code point with the `White Space` property, but Perl doesn't include `Control-K`, `U+000B LINE TABULATION` in that. Perl *does* include `\cK` in its `\v` shortcut for vertical white space, so get both flavors via `[\h\v]`, where `\h` is horizontal whitespace. Perl's `\s` matches 25 code points.

Boundaries: `\b`, `\B`

- A `\b` is a word-transition next to a word character:

```
\b is just like (?:(?<=\w)(?!w)|(?<!\w)(?=w))
\B is just like (?:(?<=\w)(?=w)|(?<!\w)(?!w))
```

- Use this idea to write your own boundaries more in line with some folks' expectations.

```
my $space_edge_left = qr{ (?:(?<=^)|(?<=[\h\v])) }x;
my $space_edge_right = qr{ (?= \z | [\h\v]) }x;
```

- Because they use `\w` to find an edge, Perl's `\b` and `\B` don't (currently) do the fancier edge detection specified in Unicode UAX #29 on Text Boundaries. See, however, the CPAN module `Unicode::LineBreak`.
- That module also comes with `Unicode::GCString`, useful for counting graphemes and especially for determining print widths.

New Regex Charclass Modifiers

- Since v5.14, Perl regexes accept new /dual "Highlander" modifiers to change how the character classes just discussed plus the old `[[:POSIX:]]` classes behave.
 - `/u` uses Unicode rules for `\w` & *c*. This is the new default under the `unicode_strings` feature. *Goodbye, Unicode Bug!*
 - `/d` uses dodgy, dual ASCII/Unicode rules; this was the old default, and still is without something in the surrounding context changing it. Avoid it.
 - `/l` (attempts to) use current POSIX locale's `ctype` rules (unreliable).
 - `/a` uses ASCII rules, with Unicode casefolding under `/i`.
 - `/aa` like `/a`, but without Unicode casefolding under `/i`.
- You inherit them from the surrounding scope, so almost never have to use them yourself. A `use v5.14` suffices to enable `/u` on all your patterns.
- The `re` pragma has been extended to accept */modifiers*:

```
use re "/msx"; # turn all those all on by default
use re "/a"; # ASCII character classes
```

Unicode Properties

- Every code point, even those not yet defined, has several and often many properties that it tests true for; minimally, *exactly one* of `\pL` `\pM` `\pN` `\pS` `\pP` `\pZ` `\pC` applies, which are abbreviations of *General Categories*.
- Detect a code point *with* a given property *NAME* and *VALUE* in a regex using the `\p` notation, as in `\p{NAME=VALUE}`, or *without it* using `\P`, as in `\P{NAME=VALUE}`.
- All Unicode properties are actually two-parters like `\p{General_Category=Lowercase_Letter}`, `\p{Script=Latin}`, `\p{Math=Yes}`, or `\p{Bidi_Class=Arabic_Letter}`.
- Since those are too hard to type, boolean properties can omit the `=Yes`, and most other property names and values have shortcuts: `\p{gc=Ll}`, `\p{sc=Latin}`, `\p{Math}`, or `\p{bc=AL}`. All are case-insensitive.
- Like boolean properties, General Categories and Scripts shorten to single words: `\p{Ll}`, `\p{Latin}`. All but the 7 one-letter GC shortcuts require braces.

Investigating Properties

- The *perluniprops* manpage, new to v5.12, lists all properties recognized by Perl, along with each one's aliases (if any) and how many code points have that property.
- The *uniclars* and *uniprops* scripts are great for figuring out which code points have which properties.

```
$ unichars -gns '\p{Cased}' '\p{Number}'
I U+2160 GC=Nl NV=1 SC=Latin ROMAN NUMERAL ONE
II U+2161 GC=Nl NV=2 SC=Latin ROMAN NUMERAL TWO
III U+2162 GC=Nl NV=3 SC=Latin ROMAN NUMERAL THREE
IV U+2163 GC=Nl NV=4 SC=Latin ROMAN NUMERAL FOUR
V U+2164 GC=Nl NV=5 SC=Latin ROMAN NUMERAL FIVE
VI U+2165 GC=Nl NV=6 SC=Latin ROMAN NUMERAL SIX
...
```

- *unichars*, *uniprops*, *uninames*, and much more of that ilk are at:

<http://training.perl.com/scripts/>

Investigating Properties *cont...*

- Call *uniprops* to find some code point's properties:

```
$ uniprops 2162
U+2162 (III) \N{ROMAN NUMERAL THREE}
\w \pN \p{Nl}
All Any Alnum Alpha Alphabetic Assigned InNumberForms
Cased Changes_When_Casemapped CWCF Changes_When_Casemapped
CWCW Changes_When_Lowercased CWL Changes_When_NFKC_Casemapped
CWKCF Nl N Gr_Base Grapheme_Base Graph GrBase ID_Continue
IDC ID_Start IDS Latin Latn Letter_Number Number_Number_Forms
Print Upper Uppercase Word XID_Continue XIDC XID_Start XIDS
X_POSIX_Alnum X_POSIX_Alpha X_POSIX_Graph X_POSIX_Print
X_POSIX_Upper X_POSIX_Word
```

- Add `-ga` to get them all, including two-parters, like `Age=1.1 BC=L CCC=NR DT=Com DT=NonCanon EA=A GC=Nl GCB=XX HST=NA JT=U Block=Number_Forms Script=Latin LB=AI NT=Nu NV=4 IN=1.1 IN=2.0 IN=2.1 IN=3.0 IN=3.1 IN=3.2 IN=4.0 IN=4.1 IN=5.0 IN=5.1 IN=5.2 IN=6.0 SB=UP WB=LE`

Unicode Tools

- In <http://training.perl.com/scripts/>, you can find:

- Probing the UCD: *unichars*, *uninames*, *uniprops*
- For use with the *charnames* pragma: *unicore/html_alias.pl*
- Unix tool rewrites: *tcgrep*, *ucsort*, *unifmt*, *unilook*, *uniquote* (a better `cat -v` or `od`), *uniwc*
- Normalization filters and tools: *nfc*, *nfd*, *nfkc*, *nfkd*, plus *nfccheck*
- Casing and font-fun filters: *uc*, *lc*, *tc*, plus *titulate*; *unifont*, *leo*, *unicaps*, *uninarrow*, *uniwide*, *unisubs*, *unisupers*
- Test/demo code: *es-sort*, *hantest*, *havshpx*, *hypertest*, *nunez* (should be called *búsqueda-libre* but for filesystem issues)

Regex Gotchas

- Code that uses things like `tr[\000-\177][\200-\377]` is broken and wrong.
- Code that assumes `\w` contains only letters, digits, and underscores is wrong — unless you use the `/a` modifier or an enclosing scope has a

```
use re "/a";
```

- Code that assumes it can remove `Mark`s to get “ASCII” letters is evil & wrong, broken & brain-damaged, and justification for corporal if not capital punishment. See my *núñez* script for how to do “accent-insensitive” matching using `Unicode::Collate` at primary strength alone.
- Code that assumes diacritics `\p{Diacritic}` and marks `\p{Mark}` are the same thing is broken.

Regex Gotchas (*cont¹...*)

- Code that assumes all `\p{Mark}` characters take zero print columns is broken.
- Code that assumes there is any limit to how many code points in a row just one `\X` can match is wrong.
- Code that assumes `\X` can never start with a `\p{Mark}` character is wrong.

