NAME

PCRE - Perl-compatible regular expressions

DIFFERENCES BETWEEN PCRE AND PERL

This document describes the differences in the ways that PCRE and Perl handle regular expressions. The differences described here are with respect to Perl versions 5.10 and above.

1. PCRE has only a subset of Perl's Unicode support. Details of what it does have are given in the **pcreunicode** page.

2. PCRE allows repeat quantifiers only on parenthesized assertions, but they do not mean what you might think. For example, (?!a){3} does not assert that the next three characters are not "a". It just asserts that the next character is not "a" three times (in principle: PCRE optimizes this to run the assertion just once). Perl allows repeat quantifiers on other assertions such as \b, but these do not seem to have any use.

3. Capturing subpatterns that occur inside negative lookahead assertions are counted, but their entries in the offsets vector are never set. Perl sometimes (but not always) sets its numerical variables from inside negative assertions.

4. Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence \0 can be used in the pattern to represent a binary zero.

5. The following Perl escape sequences are not supported: \l, \u, \L, \U, and \N when followed by a character name or Unicode value. (\N on its own, matching a non-newline character, is supported.) In fact these are implemented by Perl's general string-handling and are not part of its pattern matching engine. If any of these are encountered by PCRE, an error is generated by default. However, if the PCRE_JAVASCRIPT_COMPAT option is set, \U and \u are interpreted as JavaScript interprets them.

6. The Perl escape sequences \p, \P, and \X are supported only if PCRE is built with Unicode character property support. The properties that can be tested with \p and \P are limited to the general category properties such as Lu and Nd, script names such as Greek or Han, and the derived properties Any and L&. PCRE does support the Cs (surrogate) property, which Perl does not; the Perl documentation says "Because Perl hides the need for the user to understand the internal representation of Unicode characters, there is no need to implement the somewhat messy concept of surrogates."

7. PCRE does support the \Q...\E escape for quoting substrings. Characters in between are treated as literals. This is slightly different from Perl in that $ and @ are also handled as literals inside the quotes. In Perl, they cause variable interpolation (but of course PCRE does not have variables). Note the

following examples:

Pattern	PCRE matches	Perl matches
\Qabc$xyz\E	abc$xyz	abc followed by the contents of $xyz
\Qabc\$xyz\E	abc\$xyz	abc\$xyz
\Qabc\E\$\Qxyz\E	abc$xyz	abc$xyz

The \Q...\E sequence is recognized both inside and outside character classes.

8. Fairly obviously, PCRE does not support the (?{code}) and (??{code}) constructions. However, there is support for recursive patterns. This is not available in Perl 5.8, but it is in Perl 5.10. Also, the PCRE "callout" feature allows an external function to be called during pattern matching. See the **pcrecallout** documentation for details.

9. Subpatterns that are called as subroutines (whether or not recursively) are always treated as atomic groups in PCRE. This is like Python, but unlike Perl. Captured values that are set outside a subroutine call can be reference from inside in PCRE, but not in Perl. There is a discussion that explains these differences in more detail in the section on recursion differences from Perl in the **pcrepattern** page.

10. If any of the backtracking control verbs are used in a subpattern that is called as a subroutine (whether or not recursively), their effect is confined to that subpattern; it does not extend to the surrounding pattern. This is not always the case in Perl. In particular, if (*THEN) is present in a group that is called as a subroutine, its action is limited to that group, even if the group does not contain any | characters. Note that such subpatterns are processed as anchored at the point where they are tested.

11. If a pattern contains more than one backtracking control verb, the first one that is backtracked onto acts. For example, in the pattern A(*COMMIT)B(*PRUNE)C a failure in B triggers (*COMMIT), but a failure in C triggers (*PRUNE). Perl’s behaviour is more complex; in many cases it is the same as PCRE, but there are examples where it differs.

12. Most backtracking verbs in assertions have their normal actions. They are not confined to the assertion.

13. There are some differences that are concerned with the settings of captured strings when part of a pattern is repeated. For example, matching "aba" against the pattern /^(a(b)?)+$/ in Perl leaves $2 unset, but in PCRE it is set to "b".

14. PCRE’s handling of duplicate subpattern numbers and duplicate subpattern names is not as general

as Perl's. This is a consequence of the fact the PCRE works internally just with numbers, using an external table to translate between numbers and names. In particular, a pattern such as (?|(?<a>A)|(?<b>B), where the two capturing parentheses have the same number but different names, is not supported, and causes an error at compile time. If it were allowed, it would not be possible to distinguish which parentheses matched, because both names map to capturing subpattern number 1. To avoid this confusing situation, an error is given at compile time.

15. Perl recognizes comments in some places that PCRE does not, for example, between the (and ? at the start of a subpattern. If the /x modifier is set, Perl allows white space between (and ? (though current Perls warn that this is deprecated) but PCRE never does, even if the PCRE_EXTENDED option is set.

16. Perl, when in warning mode, gives warnings for character classes such as [A-\d] or [a-[:digit:]]. It then treats the hyphens as literals. PCRE has no warning features, so it gives an error in these cases because they are almost certainly user mistakes.

17. In PCRE, the upper/lower case character properties Lu and Ll are not affected when case-independent matching is specified. For example, \p{Lu} always matches an upper case letter. I think Perl has changed in this respect; in the release at the time of writing (5.16), \p{Lu} and \p{Ll} match all letters, regardless of case, when case independence is specified.

18. PCRE provides some extensions to the Perl regular expression facilities. Perl 5.10 includes new features that are not in earlier versions of Perl, some of which (such as named parentheses) have been in PCRE for some time. This list is with respect to Perl 5.10:

(a) Although lookbehind assertions in PCRE must match fixed length strings, each alternative branch of a lookbehind assertion can match a different length of string. Perl requires them all to have the same length.

(b) If PCRE_DOLLAR_ENDONLY is set and PCRE_MULTILINE is not set, the $ meta-character matches only at the very end of the string.

(c) If PCRE_EXTRA is set, a backslash followed by a letter with no special meaning is faulted. Otherwise, like Perl, the backslash is quietly ignored. (Perl can be made to issue a warning.)

(d) If PCRE_UNGREEDY is set, the greediness of the repetition quantifiers is inverted, that is, by default they are not greedy, but if followed by a question mark they are.

(e) PCRE_ANCHORED can be used at matching time to force a pattern to be tried only at the first matching position in the subject string.

(f) The PCRE_NOTBOL, PCRE_NOTEOL, PCRE_NOTEMPTY, PCRE_NOTEMPTY_ATSTART, and PCRE_NO_AUTO_CAPTURE options for **pcre_exec()** have no Perl equivalents.

(g) The \R escape sequence can be restricted to match only CR, LF, or CRLF by the PCRE_BSR_ANYCRLF option.

(h) The callout facility is PCRE-specific.

(i) The partial matching facility is PCRE-specific.

(j) Patterns compiled by PCRE can be saved and re-used at a later time, even on different hosts that have the other endianness. However, this does not apply to optimized data created by the just-in-time compiler.

(k) The alternative matching functions (**pcre_dfa_exec()**, **pcre16_dfa_exec()** and **pcre32_dfa_exec()**,) match in a different way and are not Perl-compatible.

(l) PCRE recognizes some special sequences such as (*CR) at the start of a pattern that set overall options that cannot be changed within the pattern.

AUTHOR

Philip Hazel
University Computing Service
Cambridge CB2 3QH, England.

REVISION

Last updated: 10 November 2013