NAME

pcretest - a program for testing Perl-compatible regular expressions.

SYNOPSIS

pcretest [options] [input file [output file]]

pcretest was written as a test program for the PCRE regular expression library itself, but it can also be used for experimenting with regular expressions. This document describes the features of the test program; for details of the regular expressions themselves, see the **pcrepattern** documentation. For details of the PCRE library function calls and their options, see the **pcreapi** , **pcre16** and **pcre32** documentation.

The input for **pcretest** is a sequence of regular expression patterns and strings to be matched, as described below. The output shows the result of each match. Options on the command line and the patterns control PCRE options and exactly what is output.

As PCRE has evolved, it has acquired many different features, and as a result, **pcretest** now has rather a lot of obscure options for testing every possible feature. Some of these options are specifically designed for use in conjunction with the test script and data files that are distributed as part of PCRE, and are unlikely to be of use otherwise. They are all documented here, but without much justification.

INPUT DATA FORMAT

Input to **pcretest** is processed line by line, either by calling the C library's **fgets()** function, or via the **libreadline** library (see below). In Unix-like environments, **fgets()** treats any bytes other than newline as data characters. However, in some Windows environments character 26 (hex 1A) causes an immediate end of file, and no further data is read. For maximum portability, therefore, it is safest to use only ASCII characters in **pcretest** input files.

The input is processed using using C's string functions, so must not contain binary zeroes, even though in Unix-like environments, **fgets()** treats any bytes other than newline as data characters.

PCRE's 8-BIT, 16-BIT AND 32-BIT LIBRARIES

From release 8.30, two separate PCRE libraries can be built. The original one supports 8-bit character strings, whereas the newer 16-bit library supports character strings encoded in 16-bit units. From release 8.32, a third library can be built, supporting character strings encoded in 32-bit units. The **pcretest** program can be used to test all three libraries. However, it is itself still an 8-bit program, reading 8-bit input and writing 8-bit output. When testing the 16-bit or 32-bit library, the patterns and data strings are converted to 16- or 32-bit format before being passed to the PCRE library functions. Results are converted to 8-bit for output.

References to functions and structures of the form **pcre[16|32]_xx** below mean "**pcre_xx** when using the 8-bit library, **pcre16_xx** when using the 16-bit library, or **pcre32_xx** when using the 32-bit library".

COMMAND LINE OPTIONS

-8 If the 8-bit library has been built, this option causes it to be used (this is the default). If the 8-bit library has not been built, this option causes an error.

-16 If the 16-bit library has been built, this option causes it to be used. If only the 16-bit library has been built, this is the default. If the 16-bit library has not been built, this option causes an error.

-32 If the 32-bit library has been built, this option causes it to be used. If only the 32-bit library has been built, this is the default. If the 32-bit library has not been built, this option causes an error.

-b Behave as if each pattern has the **/B** (show byte code) modifier; the internal form is output after compilation.

-C Output the version number of the PCRE library, and all available information about the optional features that are included, and then exit with zero exit code. All other options are ignored.

-C *option* Output information about a specific build-time option, then exit. This functionality is intended for use in scripts such as **RunTest**. The following options output the value and set the exit code as indicated:

ebcdic-nl the code for LF (= NL) in an EBCDIC environment:
0x15 or 0x25
0 if used in an ASCII environment
exit code is always 0
linksize the configured internal link size (2, 3, or 4)
exit code is set to the link size
newline the default newline setting:
CR, LF, CRLF, ANYCRLF, or ANY
exit code is always 0
bsr the default setting for what \R matches:
ANYCRLF or ANY
exit code is always 0

The following options output 1 for true or 0 for false, and set the exit code to the same

value:

```
ebcdic     compiled for an EBCDIC environment
jit        just-in-time support is available
pcre16     the 16-bit library was built
pcre32     the 32-bit library was built
pcre8      the 8-bit library was built
ucp        Unicode property support is available
utf        UTF-8 and/or UTF-16 and/or UTF-32 support
           is available
```

If an unknown option is given, an error message is output; the exit code is 0.

-d Behave as if each pattern has the **/D** (debug) modifier; the internal form and information about the compiled pattern is output after compilation; **-d** is equivalent to **-b -i**.

-dfa Behave as if each data line contains the \D escape sequence; this causes the alternative matching function, **pcre[16|32]_dfa_exec()**, to be used instead of the standard **pcre[16|32]_exec()** function (more detail is given below).

-help Output a brief summary these options and then exit.

-i Behave as if each pattern has the **/I** modifier; information about the compiled pattern is given after compilation.

-M Behave as if each data line contains the \M escape sequence; this causes PCRE to discover the minimum MATCH_LIMIT and MATCH_LIMIT_RECURSION settings by calling **pcre[16|32]_exec()** repeatedly with different limits.

-m Output the size of each compiled pattern after it has been compiled. This is equivalent to adding **/M** to each regular expression. The size is given in bytes for both libraries.

-O Behave as if each pattern has the **/O** modifier, that is disable auto-possessification for all patterns.

-o *osize* Set the number of elements in the output vector that is used when calling **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()** to be *osize*. The default value is 45, which is enough for 14 capturing subexpressions for **pcre[16|32]_exec()** or 22 different matches for **pcre[16|32]_dfa_exec()**. The vector size can be changed for individual matching calls by including \O in the data line (see below).

-p Behave as if each pattern has the **/P** modifier; the POSIX wrapper API is used to call PCRE. None of the other options has any effect when **-p** is set. This option can be used only with the 8-bit library.

-q Do not output the version number of **pcretest** at the start of execution.

-S *size* On Unix-like systems, set the size of the run-time stack to *size* megabytes.

-s or **-s+** Behave as if each pattern has the **/S** modifier; in other words, force each pattern to be studied. If **-s+** is used, all the JIT compile options are passed to **pcre[16|32]_study()**, causing just-in-time optimization to be set up if it is available, for both full and partial matching. Specific JIT compile options can be selected by following **-s+** with a digit in the range 1 to 7, which selects the JIT compile modes as follows:

1 normal match only
2 soft partial match only
3 normal match and soft partial match
4 hard partial match only
6 soft and hard partial match
7 all three modes (default)

If **-s++** is used instead of **-s+** (with or without a following digit), the text "(JIT)" is added to the first output line after a match or no match when JIT-compiled code was actually used.

Note that there are pattern options that can override **-s**, either specifying no studying at all, or suppressing JIT compilation.

If the **/I** or **/D** option is present on a pattern (requesting output about the compiled pattern), information about the result of studying is not included when studying is caused only by **-s** and neither **-i** nor **-d** is present on the command line. This behaviour means that the output from tests that are run with and without **-s** should be identical, except when options that output information about the actual running of a match are set.

The **-M**, **-t**, and **-tm** options, which give information about resources used, are likely to produce different output with and without **-s**. Output may also differ if the **/C** option is present on an individual pattern. This uses callouts to trace the the matching process, and this may be different between studied and non-studied patterns. If the pattern contains (*MARK) items there may also be differences, for the same reason. The **-s** command line option can be overridden for specific patterns that should never be studied (see the **/S**

pattern modifier below).

-t Run each compile, study, and match many times with a timer, and output the resulting times per compile, study, or match (in milliseconds). Do not set **-m** with **-t**, because you will then get the size output a zillion times, and the timing will be distorted. You can control the number of iterations that are used for timing by following **-t** with a number (as a separate item on the command line). For example, "-t 1000" iterates 1000 times. The default is to iterate 500000 times.

-tm This is like **-t** except that it times only the matching phase, not the compile or study phases.

-T -TM These behave like **-t** and **-tm**, but in addition, at the end of a run, the total times for all compiles, studies, and matches are output.

DESCRIPTION

If **pcretest** is given two filename arguments, it reads from the first and writes to the second. If it is given only one filename argument, it reads from that file and writes to stdout. Otherwise, it reads from stdin and writes to stdout, and prompts for each line of input, using "re>" to prompt for regular expressions, and "data>" to prompt for data lines.

When **pcretest** is built, a configuration option can specify that it should be linked with the **libreadline** library. When this is done, if the input is from a terminal, it is read using the **readline()** function. This provides line-editing and history facilities. The output from the **-help** option states whether or not **readline()** will be used.

The program handles any number of sets of input on a single input file. Each set starts with a regular expression, and continues with any number of data lines to be matched against that pattern.

Each data line is matched separately and independently. If you want to do multi-line matches, you have to use the \n escape sequence (or \r or \r\n, etc., depending on the newline setting) in a single line of input to encode the newline sequences. There is no limit on the length of data lines; the input buffer is automatically extended if it is too small.

An empty line signals the end of the data lines, at which point a new regular expression is read. The regular expressions are given enclosed in any non-alphanumeric delimiters other than backslash, for example:

```
/(a|bc)x+yz/
```

White space before the initial delimiter is ignored. A regular expression may be continued over several

input lines, in which case the newline characters are included within it. It is possible to include the delimiter within the pattern by escaping it, for example

```
/abc\/def/
```

If you do so, the escape and the delimiter form part of the pattern, but since delimiters are always non-alphanumeric, this does not affect its interpretation. If the terminating delimiter is immediately followed by a backslash, for example,

```
/abc/\
```

then a backslash is added to the end of the pattern. This is done to provide a way of testing the error condition that arises if a pattern finishes with a backslash, because

```
/abc\/
```

is interpreted as the first line of a pattern that starts with "abc/", causing pcretest to read the next line as a continuation of the regular expression.

PATTERN MODIFIERS

A pattern may be followed by any number of modifiers, which are mostly single characters, though some of these can be qualified by further characters. Following Perl usage, these are referred to below as, for example, "the **/i** modifier", even though the delimiter of the pattern need not always be a slash, and no slash is used when writing modifiers. White space may appear between the final pattern delimiter and the first modifier, and between the modifiers themselves. For reference, here is a complete list of modifiers. They fall into several groups that are described in detail in the following sections.

/8	set UTF mode
/9	set PCRE_NEVER_UTF (locks out UTF mode)
/?	disable UTF validity check
/+	show remainder of subject after match
/=	show all captures (not just those that are set)

/A	set PCRE_ANCHORED
/B	show compiled code
/C	set PCRE_AUTO_CALLOUT
/D	same as **/B** plus **/I**
/E	set PCRE_DOLLAR_ENDONLY
/F	flip byte order in compiled pattern

/f set PCRE_FIRSTLINE
/G find all matches (shorten string)
/g find all matches (use startoffset)
/I show information about pattern
/i set PCRE_CASELESS
/J set PCRE_DUPNAMES
/K show backtracking control names
/L set locale
/M show compiled memory size
/m set PCRE_MULTILINE
/N set PCRE_NO_AUTO_CAPTURE
/O set PCRE_NO_AUTO_POSSESS
/P use the POSIX wrapper
/Q test external stack check function
/S study the pattern after compilation
/s set PCRE_DOTALL
/T select character tables
/U set PCRE_UNGREEDY
/W set PCRE_UCP
/X set PCRE_EXTRA
/x set PCRE_EXTENDED
/Y set PCRE_NO_START_OPTIMIZE
/Z don’t show lengths in **/B** output

/<any> set PCRE_NEWLINE_ANY
/<anycrlf> set PCRE_NEWLINE_ANYCRLF
/<cr> set PCRE_NEWLINE_CR
/<crlf> set PCRE_NEWLINE_CRLF
/<lf> set PCRE_NEWLINE_LF
/<bsr_anycrlf> set PCRE_BSR_ANYCRLF
/<bsr_unicode> set PCRE_BSR_UNICODE
/<JS> set PCRE_JAVASCRIPT_COMPAT

Perl-compatible modifiers

The **/i**, **/m**, **/s**, and **/x** modifiers set the PCRE_CASELESS, PCRE_MULTILINE, PCRE_DOTALL, or PCRE_EXTENDED options, respectively, when **pcre[16|32]_compile()** is called. These four modifier letters have the same effect as they do in Perl. For example:

```
/caseless/i
```

Modifiers for other PCRE options

The following table shows additional modifiers for setting PCRE compile-time options that do not correspond to anything in Perl:

/8	PCRE_UTF8) when using the 8-bit
/?	PCRE_NO_UTF8_CHECK) library
/8	PCRE_UTF16) when using the 16-bit
/?	PCRE_NO_UTF16_CHECK) library
/8	PCRE_UTF32) when using the 32-bit
/?	PCRE_NO_UTF32_CHECK) library
/9	PCRE_NEVER_UTF
/A	PCRE_ANCHORED
/C	PCRE_AUTO_CALLOUT
/E	PCRE_DOLLAR_ENDONLY
/f	PCRE_FIRSTLINE
/J	PCRE_DUPNAMES
/N	PCRE_NO_AUTO_CAPTURE
/O	PCRE_NO_AUTO_POSSESS
/U	PCRE_UNGREEDY
/W	PCRE_UCP
/X	PCRE_EXTRA
/Y	PCRE_NO_START_OPTIMIZE
/<any>	PCRE_NEWLINE_ANY
/<anycrlf>	PCRE_NEWLINE_ANYCRLF
/<cr>	PCRE_NEWLINE_CR
/<crlf>	PCRE_NEWLINE_CRLF
/<lf>	PCRE_NEWLINE_LF
/<bsr_anycrlf>	PCRE_BSR_ANYCRLF
/<bsr_unicode>	PCRE_BSR_UNICODE
/<JS>	PCRE_JAVASCRIPT_COMPAT

The modifiers that are enclosed in angle brackets are literal strings as shown, including the angle brackets, but the letters within can be in either case. This example sets multiline matching with CRLF as the line ending sequence:

```
/^abc/m<CRLF>
```

As well as turning on the PCRE_UTF8/16/32 option, the **/8** modifier causes all non-printing characters in output strings to be printed using the \x{hh...} notation. Otherwise, those less than 0x100 are output in hex without the curly brackets.

Full details of the PCRE options are given in the **pcreapi** documentation.

Finding all matches in a string

Searching for all possible matches within each subject string can be requested by the **/g** or **/G** modifier. After finding a match, PCRE is called again to search the remainder of the subject string. The difference between **/g** and **/G** is that the former uses the *startoffset* argument to **pcre[16|32]_exec()** to start searching at a new point within the entire string (which is in effect what Perl does), whereas the latter passes over a shortened substring. This makes a difference to the matching process if the pattern begins with a lookbehind assertion (including \b or \B).

If any call to **pcre[16|32]_exec()** in a **/g** or **/G** sequence matches an empty string, the next call is done with the PCRE_NOTEMPTY_ATSTART and PCRE_ANCHORED flags set in order to search for another, non-empty, match at the same point. If this second match fails, the start offset is advanced, and the normal match is retried. This imitates the way Perl handles such cases when using the **/g** modifier or the **split()** function. Normally, the start offset is advanced by one character, but if the newline convention recognizes CRLF as a newline, and the current character is CR followed by LF, an advance of two is used.

Other modifiers

There are yet more modifiers for controlling the way **pcretest** operates.

The **/+** modifier requests that as well as outputting the substring that matched the entire pattern, **pcretest** should in addition output the remainder of the subject string. This is useful for tests where the subject contains multiple copies of the same substring. If the **+** modifier appears twice, the same action is taken for captured substrings. In each case the remainder is output on the following line with a plus character following the capture number. Note that this modifier must not immediately follow the /S modifier because /S+ and /S++ have other meanings.

The **/=** modifier requests that the values of all potential captured parentheses be output after a match. By default, only those up to the highest one actually used in the match are output (corresponding to the return code from **pcre[16|32]_exec()**). Values in the offsets vector corresponding to higher numbers should be set to -1, and these are output as "<unset>". This modifier gives a way of checking that this is happening.

The **/B** modifier is a debugging feature. It requests that **pcretest** output a representation of the compiled code after compilation. Normally this information contains length and offset values; however, if **/Z** is

also present, this data is replaced by spaces. This is a special feature for use in the automatic test scripts; it ensures that the same output is generated for different internal link sizes.

The **/D** modifier is a PCRE debugging feature, and is equivalent to **/BI**, that is, both the **/B** and the **/I** modifiers.

The **/F** modifier causes **pcretest** to flip the byte order of the 2-byte and 4-byte fields in the compiled pattern. This facility is for testing the feature in PCRE that allows it to execute patterns that were compiled on a host with a different endianness. This feature is not available when the POSIX interface to PCRE is being used, that is, when the **/P** pattern modifier is specified. See also the section about saving and reloading compiled patterns below.

The **/I** modifier requests that **pcretest** output information about the compiled pattern (whether it is anchored, has a fixed first character, and so on). It does this by calling **pcre[16|32]_fullinfo()** after compiling a pattern. If the pattern is studied, the results of that are also output. In this output, the word "char" means a non-UTF character, that is, the value of a single data item (8-bit, 16-bit, or 32-bit, depending on the library that is being tested).

The **/K** modifier requests **pcretest** to show names from backtracking control verbs that are returned from calls to **pcre[16|32]_exec()**. It causes **pcretest** to create a **pcre[16|32]_extra** block if one has not already been created by a call to **pcre[16|32]_study()**, and to set the PCRE_EXTRA_MARK flag and the **mark** field within it, every time that **pcre[16|32]_exec()** is called. If the variable that the **mark** field points to is non-NULL for a match, non-match, or partial match, **pcretest** prints the string to which it points. For a match, this is shown on a line by itself, tagged with "MK:". For a non-match it is added to the message.

The **/L** modifier must be followed directly by the name of a locale, for example,

```
/pattern/Lfr_FR
```

For this reason, it must be the last modifier. The given locale is set, **pcre[16|32]_maketables()** is called to build a set of character tables for the locale, and this is then passed to **pcre[16|32]_compile()** when compiling the regular expression. Without an **/L** (or **/T**) modifier, NULL is passed as the tables pointer; that is, **/L** applies only to the expression on which it appears.

The **/M** modifier causes the size in bytes of the memory block used to hold the compiled pattern to be output. This does not include the size of the **pcre[16|32]** block; it is just the actual compiled data. If the pattern is successfully studied with the PCRE_STUDY_JIT_COMPILE option, the size of the JIT compiled code is also output.

The **/Q** modifier is used to test the use of **pcre_stack_guard**. It must be followed by '0' or '1', specifying the return code to be given from an external function that is passed to PCRE and used for stack checking during compilation (see the **pcreapi** documentation for details).

The **/S** modifier causes **pcre[16|32]_study()** to be called after the expression has been compiled, and the results used when the expression is matched. There are a number of qualifying characters that may follow **/S**. They may appear in any order.

If **/S** is followed by an exclamation mark, **pcre[16|32]_study()** is called with the PCRE_STUDY_EXTRA_NEEDED option, causing it always to return a **pcre_extra** block, even when studying discovers no useful information.

If **/S** is followed by a second S character, it suppresses studying, even if it was requested externally by the **-s** command line option. This makes it possible to specify that certain patterns are always studied, and others are never studied, independently of **-s**. This feature is used in the test files in a few cases where the output is different when the pattern is studied.

If the **/S** modifier is followed by a + character, the call to **pcre[16|32]_study()** is made with all the JIT study options, requesting just-in-time optimization support if it is available, for both normal and partial matching. If you want to restrict the JIT compiling modes, you can follow **/S+** with a digit in the range 1 to 7:

1 normal match only
2 soft partial match only
3 normal match and soft partial match
4 hard partial match only
6 soft and hard partial match
7 all three modes (default)

If **/S++** is used instead of **/S+** (with or without a following digit), the text "(JIT)" is added to the first output line after a match or no match when JIT-compiled code was actually used.

Note that there is also an independent **/+** modifier; it must not be given immediately after **/S** or **/S+** because this will be misinterpreted.

If JIT studying is successful, the compiled JIT code will automatically be used when **pcre[16|32]_exec()** is run, except when incompatible run-time options are specified. For more details, see the **pcrejit** documentation. See also the **\J** escape sequence below for a way of setting the size of the JIT stack.

Finally, if **/S** is followed by a minus character, JIT compilation is suppressed, even if it was requested externally by the **-s** command line option. This makes it possible to specify that JIT is never to be used for certain patterns.

The **/T** modifier must be followed by a single digit. It causes a specific set of built-in character tables to be passed to **pcre[16|32]_compile()**. It is used in the standard PCRE tests to check behaviour with different character tables. The digit specifies the tables as follows:

0 the default ASCII tables, as distributed in
pcre_chartables.c.dist
1 a set of tables defining ISO 8859 characters

In table 1, some characters whose codes are greater than 128 are identified as letters, digits, spaces, etc.

Using the POSIX wrapper API

The **/P** modifier causes **pcretest** to call PCRE via the POSIX wrapper API rather than its native API. This supports only the 8-bit library. When **/P** is set, the following modifiers set options for the **regcomp()** function:

/i REG_ICASE
/m REG_NEWLINE
/N REG_NOSUB
/s REG_DOTALL)
/U REG_UNGREEDY) These options are not part of
/W REG_UCP) the POSIX standard
/8 REG_UTF8)

The /+ modifier works as described above. All other modifiers are ignored.

Locking out certain modifiers

PCRE can be compiled with or without support for certain features such as UTF-8/16/32 or Unicode properties. Accordingly, the standard tests are split up into a number of different files that are selected for running depending on which features are available. When updating the tests, it is all too easy to put a new test into the wrong file by mistake; for example, to put a test that requires UTF support into a file that is used when it is not available. To help detect such mistakes as early as possible, there is a facility for locking out specific modifiers. If an input line for **pcretest** starts with the string "< forbid " the following sequence of characters is taken as a list of forbidden modifiers. For example, in the test files that must not use UTF or Unicode property support, this line appears:

< forbid 8W

This locks out the /8 and /W modifiers. An immediate error is given if they are subsequently encountered. If the character string contains < but not >, all the multi-character modifiers that begin with < are locked out. Otherwise, such modifiers must be explicitly listed, for example:

```
< forbid <JS><cr>
```

There must be a single space between < and "forbid" for this feature to be recognised. If there is not, the line is interpreted either as a request to re-load a pre-compiled pattern (see "SAVING AND RELOADING COMPILED PATTERNS" below) or, if there is a another < character, as a pattern that uses < as its delimiter.

DATA LINES

Before each data line is passed to **pcre[16|32]_exec()**, leading and trailing white space is removed, and it is then scanned for \ escapes. Some of these are pretty esoteric features, intended for checking out some of the more complicated features of PCRE. If you are just testing "ordinary" regular expressions, you probably don’t need any of these. The following escapes are recognized:

```
\a         alarm (BEL, \x07)
\b         backspace (\x08)
\e         escape (\x27)
\f         form feed (\x0c)
\n         newline (\x0a)
\qdd       set the PCRE_MATCH_LIMIT limit to dd
           (any number of digits)
\r         carriage return (\x0d)
\t         tab (\x09)
\v         vertical tab (\x0b)
\nnn       octal character (up to 3 octal digits); always
           a byte unless > 255 in UTF-8 or 16-bit or 32-bit mode
\o{dd...}  octal character (any number of octal digits}
\xhh       hexadecimal byte (up to 2 hex digits)
\x{hh...}  hexadecimal character (any number of hex digits)
\A         pass the PCRE_ANCHORED option to pcre[16|32]_exec()
           or pcre[16|32]_dfa_exec()
\B         pass the PCRE_NOTBOL option to pcre[16|32]_exec()
           or pcre[16|32]_dfa_exec()
\Cdd       call pcre[16|32]_copy_substring() for substring dd
           after a successful match (number less than 32)
\Cname     call pcre[16|32]_copy_named_substring() for substring
           "name" after a successful match (name termin-
```

ated by next non alphanumeric character)

\C+ show the current captured substrings at callout time

\C- do not supply a callout function

\C!n return 1 instead of 0 when callout number n is reached

\C!n!m return 1 instead of 0 when callout number n is reached for the nth time

\C*n pass the number n (may be negative) as callout data; this is used as the callout return value

\D use the **pcre[16|32]_dfa_exec()** match function

\F only shortest match for **pcre[16|32]_dfa_exec()**

\Gdd call pcre[16|32]_get_substring() for substring dd after a successful match (number less than 32)

\Gname call pcre[16|32]_get_named_substring() for substring "name" after a successful match (name terminated by next non-alphanumeric character)

\Jdd set up a JIT stack of dd kilobytes maximum (any number of digits)

\L call pcre[16|32]_get_substringlist() after a successful match

\M discover the minimum MATCH_LIMIT and MATCH_LIMIT_RECURSION settings

\N pass the PCRE_NOTEMPTY option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**; if used twice, pass the PCRE_NOTEMPTY_ATSTART option

\Odd set the size of the output vector passed to **pcre[16|32]_exec()** to dd (any number of digits)

\P pass the PCRE_PARTIAL_SOFT option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**; if used twice, pass the PCRE_PARTIAL_HARD option

\Qdd set the PCRE_MATCH_LIMIT_RECURSION limit to dd (any number of digits)

\R pass the PCRE_DFA_RESTART option to **pcre[16|32]_dfa_exec()**

\S output details of memory get/free calls during matching

\Y pass the PCRE_NO_START_OPTIMIZE option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\Z pass the PCRE_NOTEOL option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\? pass the PCRE_NO_UTF[8|16|32]_CHECK option to

pcre[16|32]_exec() or **pcre[16|32]_dfa_exec()**

\>dd start the match at offset dd (optional "-"; then any number of digits); this sets the *startoffset* argument for **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\<cr> pass the PCRE_NEWLINE_CR option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\<lf> pass the PCRE_NEWLINE_LF option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\<crlf> pass the PCRE_NEWLINE_CRLF option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\<anycrlf> pass the PCRE_NEWLINE_ANYCRLF option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

\<any> pass the PCRE_NEWLINE_ANY option to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**

The use of \x{hh...} is not dependent on the use of the **/8** modifier on the pattern. It is recognized always. There may be any number of hexadecimal digits inside the braces; invalid values provoke error messages.

Note that \xhh specifies one byte rather than one character in UTF-8 mode; this makes it possible to construct invalid UTF-8 sequences for testing purposes. On the other hand, \x{hh} is interpreted as a UTF-8 character in UTF-8 mode, generating more than one byte if the value is greater than 127. When testing the 8-bit library not in UTF-8 mode, \x{hh} generates one byte for values less than 256, and causes an error for greater values.

In UTF-16 mode, all 4-digit \x{hhhh} values are accepted. This makes it possible to construct invalid UTF-16 sequences for testing purposes.

In UTF-32 mode, all 4- to 8-digit \x{...} values are accepted. This makes it possible to construct invalid UTF-32 sequences for testing purposes.

The escapes that specify line ending sequences are literal strings, exactly as shown. No more than one newline setting should be present in any data line.

A backslash followed by anything else just escapes the anything else. If the very last character is a backslash, it is ignored. This gives a way of passing an empty line as data, since a real empty line terminates the data input.

The **\J** escape provides a way of setting the maximum stack size that is used by the just-in-time optimization code. It is ignored if JIT optimization is not being used. Providing a stack that is larger

than the default 32K is necessary only for very complicated patterns.

If \M is present, **pcretest** calls **pcre[16|32]_exec()** several times, with different values in the *match_limit* and *match_limit_recursion* fields of the **pcre[16|32]_extra** data structure, until it finds the minimum numbers for each parameter that allow **pcre[16|32]_exec()** to complete without error. Because this is testing a specific feature of the normal interpretive **pcre[16|32]_exec()** execution, the use of any JIT optimization that might have been set up by the **/S+** qualifier of **-s+** option is disabled.

The *match_limit* number is a measure of the amount of backtracking that takes place, and checking it out can be instructive. For most simple matches, the number is quite small, but for patterns with very large numbers of matching possibilities, it can become large very quickly with increasing length of subject string. The *match_limit_recursion* number is a measure of how much stack (or, if PCRE is compiled with NO_RECURSE, how much heap) memory is needed to complete the match attempt.

When \O is used, the value specified may be higher or lower than the size set by the **-O** command line option (or defaulted to 45); \O applies only to the call of **pcre[16|32]_exec()** for the line in which it appears.

If the **/P** modifier was present on the pattern, causing the POSIX wrapper API to be used, the only option-setting sequences that have any effect are \B, \N, and \Z, causing REG_NOTBOL, REG_NOTEMPTY, and REG_NOTEOL, respectively, to be passed to **regexec()**.

THE ALTERNATIVE MATCHING FUNCTION

By default, **pcretest** uses the standard PCRE matching function, **pcre[16|32]_exec()** to match each data line. PCRE also supports an alternative matching function, **pcre[16|32]_dfa_test()**, which operates in a different way, and has some restrictions. The differences between the two functions are described in the **pcrematching** documentation.

If a data line contains the \D escape sequence, or if the command line contains the **-dfa** option, the alternative matching function is used. This function finds all possible matches at a given point. If, however, the \F escape sequence is present in the data line, it stops after the first match is found. This is always the shortest possible match.

DEFAULT OUTPUT FROM PCRETEST

This section describes the output when the normal matching function, **pcre[16|32]_exec()**, is being used.

When a match succeeds, **pcretest** outputs the list of captured substrings that **pcre[16|32]_exec()** returns, starting with number 0 for the string that matched the whole pattern. Otherwise, it outputs "No match" when the return is PCRE_ERROR_NOMATCH, and "Partial match:" followed by the partially

matching substring when **pcre[16|32]_exec()** returns PCRE_ERROR_PARTIAL. (Note that this is the entire substring that was inspected during the partial match; it may include characters before the actual match start if a lookbehind assertion, \K, \b, or \B was involved.) For any other return, **pcretest** outputs the PCRE negative error number and a short descriptive phrase. If the error is a failed UTF string check, the offset of the start of the failing character and the reason code are also output, provided that the size of the output vector is at least two. Here is an example of an interactive **pcretest** run.

```
$ pcretest
PCRE version 8.13 2011-04-30

  re> /^abc(\d+)/
data> abc123
 0: abc123
 1: 123
data> xyz
No match
```

Unset capturing substrings that are not followed by one that is set are not returned by **pcre[16|32]_exec()**, and are not shown by **pcretest**. In the following example, there are two capturing substrings, but when the first data line is matched, the second, unset substring is not shown. An "internal" unset substring is shown as "<unset>", as for the second data line.

```
  re> /(a)|(b)/
data> a
 0: a
 1: a
data> b
 0: b
 1: <unset>
 2: b
```

If the strings contain any non-printing characters, they are output as \xhh escapes if the value is less than 256 and UTF mode is not set. Otherwise they are output as \x{hh...} escapes. See below for the definition of non-printing characters. If the pattern has the /+ modifier, the output for substring 0 is followed by the the rest of the subject string, identified by "0+" like this:

```
  re> /cat/+
data> cataract
 0: cat
 0+ aract
```

If the pattern has the **/g** or **/G** modifier, the results of successive matching attempts are output in sequence, like this:

```
  re> /\Bi(\w\w)/g
data> Mississippi
 0: iss
 1: ss
 0: iss
 1: ss
 0: ipp
 1: pp
```

"No match" is output only if the first match attempt fails. Here is an example of a failure message (the offset 4 that is specified by \>4 is past the end of the subject string):

```
  re> /xyz/
data> xyz\>4
Error -24 (bad offset value)
```

If any of the sequences **\C**, **\G**, or **\L** are present in a data line that is successfully matched, the substrings extracted by the convenience functions are output with C, G, or L after the string number instead of a colon. This is in addition to the normal full list. The string length (that is, the return from the extraction function) is given in parentheses after each string for **\C** and **\G**.

Note that whereas patterns can be continued over several lines (a plain ">" prompt is used for continuations), data lines may not. However newlines can be included in data by means of the \n escape (or \r, \r\n, etc., depending on the newline sequence setting).

OUTPUT FROM THE ALTERNATIVE MATCHING FUNCTION

When the alternative matching function, **pcre[16|32]_dfa_exec()**, is used (by means of the \D escape sequence or the **-dfa** command line option), the output consists of a list of all the matches that start at the first point in the subject where there is at least one match. For example:

```
  re> /(tang|tangerine|tan)/
data> yellow tangerine\D
 0: tangerine
 1: tang
 2: tan
```

(Using the normal matching function on this data finds only "tang".) The longest matching string is

always given first (and numbered zero). After a PCRE_ERROR_PARTIAL return, the output is "Partial match:", followed by the partially matching substring. (Note that this is the entire substring that was inspected during the partial match; it may include characters before the actual match start if a lookbehind assertion, \K, \b, or \B was involved.)

If **/g** is present on the pattern, the search for further matches resumes at the end of the longest match. For example:

```
  re> /(tang|tangerine|tan)/g
data> yellow tangerine and tangy sultana\D
 0: tangerine
 1: tang
 2: tan
 0: tang
 1: tan
 0: tan
```

Since the matching function does not support substring capture, the escape sequences that are concerned with captured substrings are not relevant.

RESTARTING AFTER A PARTIAL MATCH

When the alternative matching function has given the PCRE_ERROR_PARTIAL return, indicating that the subject partially matched the pattern, you can restart the match with additional subject data by means of the \R escape sequence. For example:

```
  re> /^\d?\d(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\d\d$/
data> 23ja\P\D
Partial match: 23ja
data> n05\R\D
 0: n05
```

For further information about partial matching, see the **pcrepartial** documentation.

CALLOUTS

If the pattern contains any callout requests, **pcretest**'s callout function is called during matching. This works with both matching functions. By default, the called function displays the callout number, the start and current positions in the text at the callout time, and the next pattern item to be tested. For example:

```
---->pqrabcdef
```

```
  0    ^ ^     \d
```

This output indicates that callout number 0 occurred for a match attempt starting at the fourth character of the subject string, when the pointer was at the seventh character of the data, and when the next pattern item was \d. Just one circumflex is output if the start and current positions are the same.

Callouts numbered 255 are assumed to be automatic callouts, inserted as a result of the **/C** pattern modifier. In this case, instead of showing the callout number, the offset in the pattern, preceded by a plus, is output. For example:

```
   re> /\d?[A-E]\*/C
 data> E*
 --->E*
  +0 ^      \d?
  +3 ^      [A-E]
  +8 ^^     \*
 +10 ^ ^
  0: E*
```

If a pattern contains (*MARK) items, an additional line is output whenever a change of latest mark is passed to the callout function. For example:

```
   re> /a(*MARK:X)bc/C
 data> abc
 --->abc
  +0 ^       a
  +1 ^^      (*MARK:X)
 +10 ^^      b
 Latest Mark: X
 +11 ^ ^     c
 +12 ^  ^
  0: abc
```

The mark changes between matching "a" and "b", but stays the same for the rest of the match, so nothing more is output. If, as a result of backtracking, the mark reverts to being unset, the text "<unset>" is output.

The callout function in **pcretest** returns zero (carry on matching) by default, but you can use a \C item in a data line (as described above) to change this and other parameters of the callout.

Inserting callouts can be helpful when using **pcretest** to check complicated regular expressions. For further information about callouts, see the **pcrecallout** documentation.

NON-PRINTING CHARACTERS

When **pcretest** is outputting text in the compiled version of a pattern, bytes other than 32-126 are always treated as non-printing characters are are therefore shown as hex escapes.

When **pcretest** is outputting text that is a matched part of a subject string, it behaves in the same way, unless a different locale has been set for the pattern (using the **/L** modifier). In this case, the **isprint()** function to distinguish printing and non-printing characters.

SAVING AND RELOADING COMPILED PATTERNS

The facilities described in this section are not available when the POSIX interface to PCRE is being used, that is, when the **/P** pattern modifier is specified.

When the POSIX interface is not in use, you can cause **pcretest** to write a compiled pattern to a file, by following the modifiers with > and a file name. For example:

```
/pattern/im >/some/file
```

See the **pcreprecompile** documentation for a discussion about saving and re-using compiled patterns. Note that if the pattern was successfully studied with JIT optimization, the JIT data cannot be saved.

The data that is written is binary. The first eight bytes are the length of the compiled pattern data followed by the length of the optional study data, each written as four bytes in big-endian order (most significant byte first). If there is no study data (either the pattern was not studied, or studying did not return any data), the second length is zero. The lengths are followed by an exact copy of the compiled pattern. If there is additional study data, this (excluding any JIT data) follows immediately after the compiled pattern. After writing the file, **pcretest** expects to read a new pattern.

A saved pattern can be reloaded into **pcretest** by specifying < and a file name instead of a pattern. There must be no space between < and the file name, which must not contain a < character, as otherwise **pcretest** will interpret the line as a pattern delimited by < characters. For example:

```
re> </some/file
Compiled pattern loaded from /some/file
No study data
```

If the pattern was previously studied with the JIT optimization, the JIT information cannot be saved and restored, and so is lost. When the pattern has been loaded, **pcretest** proceeds to read data lines in

the usual way.

You can copy a file written by **pcretest** to a different host and reload it there, even if the new host has opposite endianness to the one on which the pattern was compiled. For example, you can compile on an i86 machine and run on a SPARC machine. When a pattern is reloaded on a host with different endianness, the confirmation message is changed to:

```
Compiled pattern (byte-inverted) loaded from /some/file
```

The test suite contains some saved pre-compiled patterns with different endianness. These are reloaded using "<!" instead of just "<". This suppresses the "(byte-inverted)" text so that the output is the same on all hosts. It also forces debugging output once the pattern has been reloaded.

File names for saving and reloading can be absolute or relative, but note that the shell facility of expanding a file name that starts with a tilde (~) is not available.

The ability to save and reload files in **pcretest** is intended for testing and experimentation. It is not intended for production use because only a single pattern can be written to a file. Furthermore, there is no facility for supplying custom character tables for use with a reloaded pattern. If the original pattern was compiled with custom tables, an attempt to match a subject string using a reloaded pattern is likely to cause **pcretest** to crash. Finally, if you attempt to load a file that is not in the correct format, the result is undefined.

SEE ALSO

pcre(3), **pcre16**(3), **pcre32**(3), **pcreapi**(3), **pcrecallout**(3), **pcrejit**, **pcrematching**(3), **pcrepartial**(d), **pcrepattern**(3), **pcreprecompile**(3).

AUTHOR

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

REVISION

Last updated: 10 February 2020