NAME

PCRE - Perl-compatible regular expressions

BUILDING PCRE

PCRE is distributed with a **configure** script that can be used to build the library in Unix-like environments using the applications known as Autotools. Also in the distribution are files to support building using **CMake** instead of **configure**. The text file **README** contains general information about building with Autotools (some of which is repeated below), and also has some comments about building on various operating systems. There is a lot more information about building PCRE without using Autotools (including information about using **CMake** and building "by hand") in the text file called **NON-AUTOTOOLS-BUILD**. You should consult this file as well as the **README** file if you are building in a non-Unix-like environment.

PCRE BUILD-TIME OPTIONS

The rest of this document describes the optional features of PCRE that can be selected when the library is compiled. It assumes use of the **configure** script, where the optional features are selected or deselected by providing options to **configure** before running the **make** command. However, the same options can be selected in both Unix-like and non-Unix-like environments using the GUI facility of **cmake-gui** if you are using **CMake** instead of **configure** to build PCRE.

If you are not using Autotools or **CMake**, option selection can be done by editing the **config.h** file, or by passing parameter settings to the compiler, as described in **NON-AUTOTOOLS-BUILD**.

The complete list of options for **configure** (which includes the standard ones such as the selection of the installation directory) can be obtained by running

```
./configure --help
```

The following sections include descriptions of options whose names begin with --enable or --disable. These settings specify changes to the defaults for the **configure** command. Because of the way that **configure** works, --enable and --disable always come in pairs, so the complementary option always exists as well, but as it specifies the default, it is not described.

BUILDING 8-BIT, 16-BIT AND 32-BIT LIBRARIES

By default, a library called **libpcre** is built, containing functions that take string arguments contained in vectors of bytes, either as single-byte characters, or interpreted as UTF-8 strings. You can also build a separate library, called **libpcre16**, in which strings are contained in vectors of 16-bit data units and interpreted either as single-unit characters or UTF-16 strings, by adding

```
--enable-pcre16
```

to the **configure** command. You can also build yet another separate library, called **libpcre32**, in which strings are contained in vectors of 32-bit data units and interpreted either as single-unit characters or UTF-32 strings, by adding

--enable-pcre32

to the **configure** command. If you do not want the 8-bit library, add

--disable-pcre8

as well. At least one of the three libraries must be built. Note that the C++ and POSIX wrappers are for the 8-bit library only, and that **pcregrep** is an 8-bit program. None of these are built if you select only the 16-bit or 32-bit libraries.

BUILDING SHARED AND STATIC LIBRARIES

The Autotools PCRE building process uses **libtool** to build both shared and static libraries by default. You can suppress one of these by adding one of

--disable-shared
--disable-static

to the **configure** command, as required.

C++ SUPPORT

By default, if the 8-bit library is being built, the **configure** script will search for a C++ compiler and C++ header files. If it finds them, it automatically builds the C++ wrapper library (which supports only 8-bit strings). You can disable this by adding

--disable-cpp

to the **configure** command.

UTF-8, UTF-16 AND UTF-32 SUPPORT

To build PCRE with support for UTF Unicode character strings, add

--enable-utf

to the **configure** command. This setting applies to all three libraries, adding support for UTF-8 to the 8-bit library, support for UTF-16 to the 16-bit library, and support for UTF-32 to the to the 32-bit library. There are no separate options for enabling UTF-8, UTF-16 and UTF-32 independently because

that would allow ridiculous settings such as requesting UTF-16 support while building only the 8-bit library. It is not possible to build one library with UTF support and another without in the same configuration. (For backwards compatibility, --enable-utf8 is a synonym of --enable-utf.)

Of itself, this setting does not make PCRE treat strings as UTF-8, UTF-16 or UTF-32. As well as compiling PCRE with this option, you also have have to set the PCRE_UTF8, PCRE_UTF16 or PCRE_UTF32 option (as appropriate) when you call one of the pattern compiling functions.

If you set --enable-utf when compiling in an EBCDIC environment, PCRE expects its input to be either ASCII or UTF-8 (depending on the run-time option). It is not possible to support both EBCDIC and UTF-8 codes in the same version of the library. Consequently, --enable-utf and --enable-ebcdic are mutually exclusive.

UNICODE CHARACTER PROPERTY SUPPORT

UTF support allows the libraries to process character codepoints up to 0x10ffff in the strings that they handle. On its own, however, it does not provide any facilities for accessing the properties of such characters. If you want to be able to use the pattern escapes \P, \p, and \X, which refer to Unicode character properties, you must add

```
--enable-unicode-properties
```

to the **configure** command. This implies UTF support, even if you have not explicitly requested it.

Including Unicode property support adds around 30K of tables to the PCRE library. Only the general category properties such as *Lu* and *Nd* are supported. Details are given in the **pcrepattern** documentation.

JUST-IN-TIME COMPILER SUPPORT

Just-in-time compiler support is included in the build by specifying

```
--enable-jit
```

This support is available only for certain hardware architectures. If this option is set for an unsupported architecture, a compile time error occurs. See the **pcrejit** documentation for a discussion of JIT usage. When JIT support is enabled, pcregrep automatically makes use of it, unless you add

```
--disable-pcregrep-jit
```

to the "configure" command.

CODE VALUE OF NEWLINE

By default, PCRE interprets the linefeed (LF) character as indicating the end of a line. This is the normal newline character on Unix-like systems. You can compile PCRE to use carriage return (CR) instead, by adding

--enable-newline-is-cr

to the **configure** command. There is also a --enable-newline-is-lf option, which explicitly specifies linefeed as the newline character.

Alternatively, you can specify that line endings are to be indicated by the two character sequence CRLF. If you want this, add

--enable-newline-is-crlf

to the **configure** command. There is a fourth option, specified by

--enable-newline-is-anycrlf

which causes PCRE to recognize any of the three sequences CR, LF, or CRLF as indicating a line ending. Finally, a fifth option, specified by

--enable-newline-is-any

causes PCRE to recognize any Unicode newline sequence.

Whatever line ending convention is selected when PCRE is built can be overridden when the library functions are called. At build time it is conventional to use the standard for your operating system.

WHAT \R MATCHES

By default, the sequence \R in a pattern matches any Unicode newline sequence, whatever has been selected as the line ending sequence. If you specify

--enable-bsr-anycrlf

the default is changed so that \R matches only CR, LF, or CRLF. Whatever is selected when PCRE is built can be overridden when the library functions are called.

POSIX MALLOC USAGE

When the 8-bit library is called through the POSIX interface (see the **pcreposix** documentation),

additional working storage is required for holding the pointers to capturing substrings, because PCRE requires three integers per substring, whereas the POSIX interface provides only two. If the number of expected substrings is small, the wrapper function uses space on the stack, because this is faster than using **malloc()** for each call. The default threshold above which the stack is no longer used is 10; it can be changed by adding a setting such as

--with-posix-malloc-threshold=20

to the **configure** command.

HANDLING VERY LARGE PATTERNS

Within a compiled pattern, offset values are used to point from one part to another (for example, from an opening parenthesis to an alternation metacharacter). By default, in the 8-bit and 16-bit libraries, two-byte values are used for these offsets, leading to a maximum size for a compiled pattern of around 64K. This is sufficient to handle all but the most gigantic patterns. Nevertheless, some people do want to process truly enormous patterns, so it is possible to compile PCRE to use three-byte or four-byte offsets by adding a setting such as

--with-link-size=3

to the **configure** command. The value given must be 2, 3, or 4. For the 16-bit library, a value of 3 is rounded up to 4. In these libraries, using longer offsets slows down the operation of PCRE because it has to load additional data when handling them. For the 32-bit library the value is always 4 and cannot be overridden; the value of --with-link-size is ignored.

AVOIDING EXCESSIVE STACK USAGE

When matching with the **pcre_exec()** function, PCRE implements backtracking by making recursive calls to an internal function called **match()**. In environments where the size of the stack is limited, this can severely limit PCRE's operation. (The Unix environment does not usually suffer from this problem, but it may sometimes be necessary to increase the maximum stack size. There is a discussion in the **pcrestack** documentation.) An alternative approach to recursion that uses memory from the heap to remember data, instead of using recursive function calls, has been implemented to work round the problem of limited stack size. If you want to build a version of PCRE that works this way, add

--disable-stack-for-recursion

to the **configure** command. With this configuration, PCRE will use the **pcre_stack_malloc** and **pcre_stack_free** variables to call memory management functions. By default these point to **malloc()** and **free()**, but you can replace the pointers so that your own functions are used instead.

Separate functions are provided rather than using **pcre_malloc** and **pcre_free** because the usage is very predictable: the block sizes requested are always the same, and the blocks are always freed in reverse order. A calling program might be able to implement optimized functions that perform better than **malloc()** and **free()**. PCRE runs noticeably more slowly when built in this way. This option affects only the **pcre_exec()** function; it is not relevant for **pcre_dfa_exec()**.

LIMITING PCRE RESOURCE USAGE

Internally, PCRE has a function called **match()**, which it calls repeatedly (sometimes recursively) when matching a pattern with the **pcre_exec()** function. By controlling the maximum number of times this function may be called during a single matching operation, a limit can be placed on the resources used by a single call to **pcre_exec()**. The limit can be changed at run time, as described in the **pcreapi** documentation. The default is 10 million, but this can be changed by adding a setting such as

```
--with-match-limit=500000
```

to the **configure** command. This setting has no effect on the **pcre_dfa_exec()** matching function.

In some environments it is desirable to limit the depth of recursive calls of **match()** more strictly than the total number of calls, in order to restrict the maximum amount of stack (or heap, if --disable-stack-for-recursion is specified) that is used. A second limit controls this; it defaults to the value that is set for --with-match-limit, which imposes no additional constraints. However, you can set a lower limit by adding, for example,

```
--with-match-limit-recursion=10000
```

to the **configure** command. This value can also be overridden at run time.

CREATING CHARACTER TABLES AT BUILD TIME

PCRE uses fixed tables for processing characters whose code values are less than 256. By default, PCRE is built with a set of tables that are distributed in the file *pcre_chartables.c.dist*. These tables are for ASCII codes only. If you add

```
--enable-rebuild-chartables
```

to the **configure** command, the distributed tables are no longer used. Instead, a program called **dftables** is compiled and run. This outputs the source for new set of tables, created in the default locale of your C run-time system. (This method of replacing the tables does not work if you are cross compiling, because **dftables** is run on the local host. If you need to create alternative tables when cross compiling, you will have to do so "by hand".)

USING EBCDIC CODE

PCRE assumes by default that it will run in an environment where the character code is ASCII (or Unicode, which is a superset of ASCII). This is the case for most computer operating systems. PCRE can, however, be compiled to run in an EBCDIC environment by adding

```
--enable-ebcdic
```

to the **configure** command. This setting implies --enable-rebuild-chartables. You should only use it if you know that you are in an EBCDIC environment (for example, an IBM mainframe operating system). The --enable-ebcdic option is incompatible with --enable-utf.

The EBCDIC character that corresponds to an ASCII LF is assumed to have the value 0x15 by default. However, in some EBCDIC environments, 0x25 is used. In such an environment you should use

```
--enable-ebcdic-nl25
```

as well as, or instead of, --enable-ebcdic. The EBCDIC character for CR has the same value as in ASCII, namely, 0x0d. Whichever of 0x15 and 0x25 is *not* chosen as LF is made to correspond to the Unicode NEL character (which, in Unicode, is 0x85).

The options that select newline behaviour, such as --enable-newline-is-cr, and equivalent run-time options, refer to these character values in an EBCDIC environment.

PCREGREP OPTIONS FOR COMPRESSED FILE SUPPORT

By default, **pcregrep** reads all files as plain text. You can build it so that it recognizes files whose names end in **.gz** or **.bz2**, and reads them with **libz** or **libbz2**, respectively, by adding one or both of

```
--enable-pcregrep-libz
--enable-pcregrep-libbz2
```

to the **configure** command. These options naturally require that the relevant libraries are installed on your system. Configuration will fail if they are not.

PCREGREP BUFFER SIZE

pcregrep uses an internal buffer to hold a "window" on the file it is scanning, in order to be able to output "before" and "after" lines when it finds a match. The size of the buffer is controlled by a parameter whose default value is 20K. The buffer itself is three times this size, but because of the way it is used for holding "before" lines, the longest line that is guaranteed to be processable is the parameter size. You can change the default parameter value by adding, for example,

--with-pcregrep-bufsize=50K

to the **configure** command. The caller of **pcregrep** can, however, override this value by specifying a run-time option.

PCRETEST OPTION FOR LIBREADLINE SUPPORT

If you add

--enable-pcretest-libreadline

to the **configure** command, **pcretest** is linked with the **libreadline** library, and when its input is from a terminal, it reads it using the **readline()** function. This provides line-editing and history facilities. Note that **libreadline** is GPL-licensed, so if you distribute a binary of **pcretest** linked in this way, there may be licensing issues.

Setting this option causes the **-lreadline** option to be added to the **pcretest** build. In many operating environments with a sytem-installed **libreadline** this is sufficient. However, in some environments (e.g. if an unmodified distribution version of readline is in use), some extra configuration may be necessary. The INSTALL file for **libreadline** says this:

"Readline uses the termcap functions, but does not link with the
termcap or curses library itself, allowing applications which link
with readline the to choose an appropriate library."

If your environment has not been set up so that an appropriate library is automatically included, you may need to add something like

LIBS="-ncurses"

immediately before the **configure** command.

DEBUGGING WITH VALGRIND SUPPORT

By adding the

--enable-valgrind

option to to the **configure** command, PCRE will use valgrind annotations to mark certain memory regions as unaddressable. This allows it to detect invalid memory accesses, and is mostly useful for debugging PCRE itself.

CODE COVERAGE REPORTING

If your C compiler is gcc, you can build a version of PCRE that can generate a code coverage report for its test suite. To enable this, you must install **lcov** version 1.6 or above. Then specify

```
--enable-coverage
```

to the **configure** command and build PCRE in the usual way.

Note that using **ccache** (a caching C compiler) is incompatible with code coverage reporting. If you have configured **ccache** to run automatically on your system, you must set the environment variable

```
CCACHE_DISABLE=1
```

before running **make** to build PCRE, so that **ccache** is not used.

When --enable-coverage is used, the following addition targets are added to the *Makefile*:

```
make coverage
```

This creates a fresh coverage report for the PCRE test suite. It is equivalent to running "make coverage-reset", "make coverage-baseline", "make check", and then "make coverage-report".

```
make coverage-reset
```

This zeroes the coverage counters, but does nothing else.

```
make coverage-baseline
```

This captures baseline coverage information.

```
make coverage-report
```

This creates the coverage report.

```
make coverage-clean-report
```

This removes the generated coverage report without cleaning the coverage data itself.

```
make coverage-clean-data
```

This removes the captured coverage data without removing the coverage files created at compile time (*.gcno).

```
make coverage-clean
```

This cleans all coverage data including the generated coverage report. For more information about code coverage, see the **gcov** and **lcov** documentation.

SEE ALSO

pcreapi(3), **pcre16**, **pcre32**, **pcre_config**(3).

AUTHOR

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

REVISION

Last updated: 12 May 2013