

NAME

perlref - Perl Regular Expressions Reference

DESCRIPTION

This is a quick reference to Perl's regular expressions. For full information see *perlre* and *perlop*, as well as the *SEE ALSO* section in this document.

OPERATORS

`=~` determines to which variable the regex is applied.
In its absence, `$_` is used.

```
$var =~ /foo/;
```

`!~` determines to which variable the regex is applied,
and negates the result of the match; it returns
false if the match succeeds, and true if it fails.

```
$var !~ /foo/;
```

`m/pattern/igmsoxc` searches a string for a pattern match,
applying the given options.

```
i  case-Insensitive
g  Global - all occurrences
m  Multiline mode - ^ and $ match internal lines
s  match as a Single line - . matches \n
o  compile pattern Once
x  eXtended legibility - free whitespace and comments
c  don't reset pos on failed matches when using /g
```

If 'pattern' is an empty string, the last I<successfully> matched
regex is used. Delimiters other than `'/'` may be used for both this
operator and the following ones.

`qr/pattern/imsox` lets you store a regex in a variable,
or pass one around. Modifiers as for `m//` and are stored
within the regex.

`s/pattern/replacement/igmsoux` substitutes matches of
'pattern' with 'replacement'. Modifiers as for `m//`
with one addition:

```
e  Evaluate replacement as an expression
```

'e' may be specified multiple times. 'replacement' is interpreted
as a double quoted string unless a single-quote (') is the delimiter.

`?pattern?` is like `m/pattern/` but matches only once. No alternate
delimiters can be used. Must be reset with `L<reset|perlfunc/reset>`.

SYNTAX

<code>\</code>	Escapes the character immediately following it
<code>.</code>	Matches any single character except a newline (unless <code>/s</code> is used)
<code>^</code>	Matches at the beginning of the string (or line, if <code>/m</code> is used)
<code>\$</code>	Matches at the end of the string (or line, if <code>/m</code> is used)
<code>*</code>	Matches the preceding element 0 or more times
<code>+</code>	Matches the preceding element 1 or more times
<code>?</code>	Matches the preceding element 0 or 1 times
<code>{...}</code>	Specifies a range of occurrences for the element preceding it
<code>[...]</code>	Matches any one of the characters contained within the brackets
<code>(...)</code>	Groups subexpressions for capturing to <code>\$1</code> , <code>\$2</code> ...
<code>(?:...)</code>	Groups subexpressions without capturing (cluster)
<code> </code>	Matches either the subexpression preceding or following it
<code>\1</code> , <code>\2</code> ...	The text from the Nth group

ESCAPE SEQUENCES

These work as in normal strings.

<code>\a</code>	Alarm (beep)
<code>\e</code>	Escape
<code>\f</code>	Formfeed
<code>\n</code>	Newline
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\037</code>	Any octal ASCII value
<code>\x7f</code>	Any hexadecimal ASCII value
<code>\x{263a}</code>	A wide hexadecimal value
<code>\cx</code>	Control-x
<code>\N{name}</code>	A named character
<code>\l</code>	Lowercase next character
<code>\u</code>	Titlecase next character
<code>\L</code>	Lowercase until <code>\E</code>
<code>\U</code>	Uppercase until <code>\E</code>
<code>\Q</code>	Disable pattern metacharacters until <code>\E</code>
<code>\E</code>	End case modification

For Titlecase, see *Titlecase*.

This one works differently from normal strings:

<code>\b</code>	An assertion, not backspace, except in a character class
-----------------	----------------------------------------------------------

CHARACTER CLASSES

<code>[amy]</code>	Match 'a', 'm' or 'y'
<code>[f-j]</code>	Dash specifies "range"
<code>[f-j-]</code>	Dash escaped or at start or end means 'dash'
<code>[^f-j]</code>	Caret indicates "match any character _except_ these"

The following sequences work within or without a character class. The first six are locale aware, all are Unicode aware. The default character class equivalent are given. See *perllocale* and *perlunicode* for details.

<code>\d</code>	A digit	<code>[0-9]</code>
-----------------	---------	--------------------

<code>\D</code>	A nondigit	<code>[^0-9]</code>
<code>\w</code>	A word character	<code>[a-zA-Z0-9_]</code>
<code>\W</code>	A non-word character	<code>[^a-zA-Z0-9_]</code>
<code>\s</code>	A whitespace character	<code>[\t\n\r\f]</code>
<code>\S</code>	A non-whitespace character	<code>[^ \t\n\r\f]</code>
<code>\C</code>	Match a byte (with Unicode, <code>'.'</code> matches a character)	
<code>\pP</code>	Match P-named (Unicode) property	
<code>\p{...}</code>	Match Unicode property with long name	
<code>\PP</code>	Match non-P	
<code>\P{...}</code>	Match lack of Unicode property with long name	
<code>\X</code>	Match extended unicode sequence	

POSIX character classes and their Unicode and Perl equivalents:

<code>alnum</code>	<code>IsAlnum</code>	Alphanumeric
<code>alpha</code>	<code>IsAlpha</code>	Alphabetic
<code>ascii</code>	<code>IsASCII</code>	Any ASCII char
<code>blank</code>	<code>IsSpace [\t]</code>	Horizontal whitespace (GNU extension)
<code>cntrl</code>	<code>IsCntrl</code>	Control characters
<code>digit</code>	<code>IsDigit \d</code>	Digits
<code>graph</code>	<code>IsGraph</code>	Alphanumeric and punctuation
<code>lower</code>	<code>IsLower</code>	Lowercase chars (locale and Unicode aware)
<code>print</code>	<code>IsPrint</code>	Alphanumeric, punct, and space
<code>punct</code>	<code>IsPunct</code>	Punctuation
<code>space</code>	<code>IsSpace [\s\ck]</code>	Whitespace
	<code>IsSpacePerl \s</code>	Perl's whitespace definition
<code>upper</code>	<code>IsUpper</code>	Uppercase chars (locale and Unicode aware)
<code>word</code>	<code>IsWord \w</code>	Alphanumeric plus <code>_</code> (Perl extension)
<code>xdigit</code>	<code>IsXDigit [0-9A-Fa-f]</code>	Hexadecimal digit

Within a character class:

POSIX	traditional	Unicode
<code>[:digit:]</code>	<code>\d</code>	<code>\p{IsDigit}</code>
<code>[:^digit:]</code>	<code>\D</code>	<code>\P{IsDigit}</code>

ANCHORS

All are zero-width assertions.

<code>^</code>	Match string start (or line, if <code>/m</code> is used)
<code>\$</code>	Match string end (or line, if <code>/m</code> is used) or before newline
<code>\b</code>	Match word boundary (between <code>\w</code> and <code>\W</code>)
<code>\B</code>	Match except at word boundary (between <code>\w</code> and <code>\w</code> or <code>\W</code> and <code>\W</code>)
<code>\A</code>	Match string start (regardless of <code>/m</code>)
<code>\Z</code>	Match string end (before optional newline)
<code>\z</code>	Match absolute string end
<code>\G</code>	Match where previous <code>m//g</code> left off

QUANTIFIERS

Quantifiers are greedy by default -- match the **longest** leftmost.

Maximal	Minimal	Allowed range
<code>{n,m}</code>	<code>{n,m}?</code>	Must occur at least <code>n</code> times but no more than <code>m</code> times

<code>{n,}</code>	<code>{n,}? </code>	Must occur at least n times
<code>{n}</code>	<code>{n}? </code>	Must occur exactly n times
<code>*</code>	<code>*? </code>	0 or more times (same as <code>{0,}</code>)
<code>+</code>	<code>+? </code>	1 or more times (same as <code>{1,}</code>)
<code>?</code>	<code>?? </code>	0 or 1 time (same as <code>{0,1}</code>)

There is no quantifier `{,n}` -- that gets understood as a literal string.

EXTENDED CONSTRUCTS

<code>(?#text)</code>	A comment
<code>(?imxs-imsx:...)</code>	Enable/disable option (as per <code>m//</code> modifiers)
<code>(?=...)</code>	Zero-width positive lookahead assertion
<code>(?!...)</code>	Zero-width negative lookahead assertion
<code>(?<=...)</code>	Zero-width positive lookbehind assertion
<code>(?<!...)</code>	Zero-width negative lookbehind assertion
<code>(?>...)</code>	Grab what we can, prohibit backtracking
<code>(?{ code })</code>	Embedded code, return value becomes <code>\$^R</code>
<code>(??{ code })</code>	Dynamic regex, return value used as regex
<code>(?(cond)yes no)</code>	cond being integer corresponding to capturing parens
<code>(?(cond)yes)</code>	or a lookaround/eval zero-width assertion

VARIABLES

<code>\$_</code>	Default variable for operators to use
<code>\$*</code>	Enable multiline matching (deprecated; not in 5.9.0 or later)
<code>\$&</code>	Entire matched string
<code>\$'</code>	Everything prior to matched string
<code>\$'</code>	Everything after to matched string

The use of those last three will slow down **all** regex use within your program. Consult *perlvar* for `@LAST_MATCH_START` to see equivalent expressions that won't cause slow down. See also *Devel::SawAmpersand*.

<code>\$1, \$2 ...</code>	hold the Xth captured expr
<code>\$+</code>	Last parenthesized pattern match
<code>^N</code>	Holds the most recently closed capture
<code>^R</code>	Holds the result of the last <code>(?{...})</code> expr
<code>@-</code>	Offsets of starts of groups. <code>\$-[0]</code> holds start of whole match
<code>@+</code>	Offsets of ends of groups. <code>\$+[0]</code> holds end of whole match

Captured groups are numbered according to their *opening* paren.

FUNCTIONS

<code>lc</code>	Lowercase a string
<code>lcfirst</code>	Lowercase first char of a string
<code>uc</code>	Uppercase a string
<code>ucfirst</code>	Titlecase first char of a string
<code>pos</code>	Return or set current match position
<code>quotemeta</code>	Quote metacharacters
<code>reset</code>	Reset <code>?pattern?</code> status
<code>study</code>	Analyze string for optimizing matching

`split` Use regex to split a string into parts

The first four of these are like the escape sequences `\L`, `\l`, `\U`, and `\u`. For Titlecase, see *Titlecase*.

TERMINOLOGY

Titlecase

Unicode concept which most often is equal to uppercase, but for certain characters like the German "sharp s" there is a difference.

AUTHOR

Iain Truskett.

This document may be distributed under the same terms as Perl itself.

SEE ALSO

- *perlretut* for a tutorial on regular expressions.
- *perlrequick* for a rapid tutorial.
- *perlre* for more details.
- *perlvar* for details on the variables.
- *perlop* for details on the operators.
- *perlfunc* for details on the functions.
- *perlfaq6* for FAQs on regular expressions.
- The *re* module to alter behaviour and aid debugging.
- "*Debugging regular expressions*" in *perldebug*
- *perluniintro*, *perlunicode*, *chardnames* and *locale* for details on regexes and internationalisation.
- *Mastering Regular Expressions* by Jeffrey Friedl (<http://regex.info/>) for a thorough grounding and reference on the topic.

THANKS

David P.C. Wollmann, Richard Soderberg, Sean M. Burke, Tom Christiansen, Jim Cromie, and Jeffrey Goff for useful advice.