Perl Regular Expression Quick Reference Card

Revision 0.1 (draft) for Perl 5.8.5
Iain Truskett (formatting by Andrew Ford) refcards.com™

This is a quick reference to Perl's regular expressions. For full information see the *perlre* and *perlop* manual pages.

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-**i**nsensitive
- `g` **g**lobal – all occurrences
- `m` **m**ultiline mode – `^` and `$` match internal lines
- `s` match as a **s**ingle line – `.` matches `\n`
- `o` compile pattern **o**nce
- `x` e**x**tended legibility – free whitespace and comments
- `c` don't reset pos on failed matches when using `/g`

If *`pattern`* is an empty string, the last 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`*`/igmsoxe`
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 `reset`.

Syntax

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

Escape sequences

These work as in normal strings.

`\a`	Alarm (beep)
`\e`	Escape
`\f`	Formfeed
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\038`	Any octal ASCII value
`\x7f`	Any hexadecimal ASCII value
`\x{263a}`	A wide hexadecimal value
`\cx`	Control-x
`\N{name}`	A named character
`\l`	Lowercase next character
`\u`	Titlecase next character
`\L`	Lowercase until `\E`
`\U`	Uppercase until `\E`
`\Q`	Disable pattern metacharacters until `\E`
`\E`	End case modification

This one works differently from normal strings:

`\b`	An assertion, not backspace, except in a character class

Character classes

`[amy]`	Match 'a', 'm' or 'y'
`[f-j]`	Dash specifies *range*
`[f-j-]`	Dash escaped or at start or end means 'dash'
`[^f-j]`	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 the perllocale and perlunicode man pages for details.

`\d`	A digit	`[0-9]`
`\D`	A nondigit	`[^0-9]`
`\w`	A word character	`[a-zA-Z0-9_]`
`\W`	A non-word character	`[^a-zA-Z0-9_]`
`\s`	A whitespace character	`[ \t\n\r\f]`
`\S`	A non-whitespace character	`[^ \t\n\r\f]`

`\C`	Match a byte (with Unicode, '`.`' matches a character)
`\pP`	Match P-named (Unicode) property
`\p{...}`	Match Unicode property with long name
`\PP`	Match non-P
`\P{...}`	Match lack of Unicode property with long name
`\X`	Match extended unicode sequence

POSIX character classes and their Unicode and Perl equivalents:

`alnum`	`IsAlnum`
`alpha`	`IsAlpha`
`ascii`	`IsASCII`
`blank`	`IsSpace`
`cntrl`	`IsCntrl`
`digit`	`IsDigit`
`graph`	`IsGraph`
`lower`	`IsLower`
`print`	`IsPrint`
`punct`	`IsPunct`
`space`	`IsSpace`
	`IsSpacePerl`
`upper`	`IsUpper`
`word`	`IsWord`
`xdigit`	`IsXDigit`

Within a character class:

POSIX	traditional	Unicode
`[:digit:]`	`\d`	`\p{IsDigit}`
`[:^digit:]`	`\D`	`\P{IsDigit}`

Anchors

All are zero-width assertions.

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

Quantifiers

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

Maximal	Minimal	Allowed range
`{n,m}`	`{n,m}?`	Must occur at least n times but no more than m times
`{n,}`	`{n,}?`	Must occur at least n times
`{n}`	`{n}?`	Must occur exactly n times
`*`	`*?`	0 or more times (same as `{0,}`)
`+`	`+?`	1 or more times (same as `{1,}`)
`?`	`??`	0 or 1 time (same as `{0,1}`)

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

Extended constructs

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

Variables

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

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

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

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

Functions

`lc`	Lowercase a string
`lcfirst`	Lowercase first char of a string
`uc`	Uppercase a string
`ucfirst`	Titlecase first char of a string
`pos`	Return or set current match position
`quotemeta`	Quote metacharacters
`reset`	Reset `?`*pattern*`?` status
`study`	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 below.

Terminology

Titlecase

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

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*, *charnames* 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.

Authors

This card was created by Andrew Ford.

The original document (`perlreref.pod`) is part of the standard Perl distribution. It was written by Iain Truskett, with thanks to David P.C. Wollmann, Richard Soderberg, Sean M. Burke, Tom Christiansen, Jim Cromie, and Jeffrey Goff for useful advice.

Perl Regular Expression Quick Reference Card
Revision 0.1 (draft) for Perl version Perl 5.8.5 [July 2005]
A refcards.com™ quick reference card.
refcards.com is a trademark of Ford & Mason Ltd.
Published by Ford & Mason Ltd.