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, true if it fails.
$var !" /foo/;
m/pattern//gmsoxc

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 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/gmsox

substitutes matches of pattern with replacement. Modifiers as for m// with one addition: '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 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...
\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
\\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 \x
\\L Uppercase until \x
\\\Disable pattern metacharacters until \x
\\\ 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 perloca and perlucode man pages for details.

POSIX character classes and their Unicode and Perl equivalents:

Within a character class:

POSIX traditional Unicode

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
# Match word boundary (between \w and \W)
$\ 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)
\b 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
(?imx:-imx:...)  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
(?{(code })  Embedded code, return value becomes $ˆR
(?{(code )}  Dynamic regex, return value used as regex
(?{(cond )}yes|no)  cond being integer corresponding to capturing paren
(?{(cond)yes})  or a lookahead/eval zero-width assertion

Variables
$_  Default variable for operators to use
$*  Enable multiline matching (deprecated; not in 5.9.0 or later)
$n  Entire matched string
$s  Everything prior to matched string
$s'  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
θ-  Offsets of starts of groups. $-[0] holds start of whole match
θ+  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.
• perlreu for more details.
• perlvar for details on the variables.
• perllop for details on the operators.
• perlmeta for details on the functions.
• perlfaq6 for FAQs on regular expressions.
• perldebug
• 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 (perlrecref.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.