NAME

perlreref - 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/msixpogc` searches a string for a pattern match, applying the given options.

```
    m  Multiline mode - ^ and $ match internal lines
    s  match as a Single line - . matches \n
    i  case-Insensitive
    x  eXtended legibility - free whitespace and comments
    p  Preserve a copy of the matched string -
       ${^PREMATCH}, ${^MATCH}, ${^POSTMATCH} will be defined.
    o  compile pattern Once
    g  Global - all occurrences
    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. The leading `m` can be omitted if the delimiter is '/'.

`qr/pattern/msixpo` 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/msixpogce` 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, \3 ...           Matches the text from the Nth group
\g1 or \g{1}, \g2 ...    Matches the text from the Nth group
\g-1 or \g{-1}, \g-2 ... Matches the text from the Nth previous group
\g{name}     Named backreference
\k<name>     Named backreference
\k'name'     Named backreference
(?P=name)    Named backreference (python syntax)
```

ESCAPE SEQUENCES

These work as in normal strings.

```
\a       Alarm (beep)
\e       Escape
\f       Formfeed
\n       Newline
\r       Carriage return
\t       Tab
\037     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 modification
```

For Titlecase, see *Titlecase*.

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. See *perllocale* and *perlunicode* for details.

```
\d      A digit
\D      A nondigit
\w      A word character
\W      A non-word character
\s      A whitespace character
\S      A non-whitespace character
```

```
\h      An horizontal white space
\H      A non horizontal white space
\v      A vertical white space
\V      A non vertical white space
\R      A generic newline           (?>\v|\x0D\x0A)

\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 combining character sequence
```

POSIX character classes and their Unicode and Perl equivalents:

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

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

\K Keep the stuff left of the \K, don't include it in $&
```

QUANTIFIERS

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

```
Maximal Minimal Possessive Allowed range
```

```
    ------- ------- ---------- -------------
    {n,m}   {n,m}?  {n,m}+     Must occur at least n times
                               but no more than m times
    {n,}    {n,}?   {n,}+      Must occur at least n times
    {n}     {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})
```

The possessive forms (new in Perl 5.10) prevent backtracking: what gets matched by a pattern with a possessive quantifier will not be backtracked into, even if that causes the whole match to fail.

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

EXTENDED CONSTRUCTS

```
    (?#text)          A comment
    (?:...)           Groups subexpressions without capturing (cluster)
    (?pimsx-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
    (?|...)           Branch reset
    (?<name>...)      Named capture
    (?'name'...)      Named capture
    (?P<name>...)     Named capture (python syntax)
    (?{ code })       Embedded code, return value becomes $^R
    (??{ code })      Dynamic regex, return value used as regex
    (?N)              Recurse into subpattern number N
    (?-N), (?+N)      Recurse into Nth previous/next subpattern
    (?R), (?0)        Recurse at the beginning of the whole pattern
    (?&name)          Recurse into a named subpattern
    (?P>name)         Recurse into a named subpattern (python syntax)
    (?(cond)yes|no)
    (?(cond)yes)      Conditional expression, where "cond" can be:
                      (N)       subpattern N has matched something
                      (<name>)  named subpattern has matched something
                      ('name')  named subpattern has matched something
                      (?{code}) code condition
                      (R)       true if recursing
                      (RN)      true if recursing into Nth subpattern
                      (R&name)  true if recursing into named subpattern
                      (DEFINE)  always false, no no-pattern allowed
```

VARIABLES

```
    $_    Default variable for operators to use

    $`    Everything prior to matched string
    $&    Entire matched string
    $'    Everything after to matched string

    ${^PREMATCH}   Everything prior to matched string
    ${^MATCH}      Entire matched string
    ${^POSTMATCH}  Everything after to matched string
```

The use of `` $` ``, `$&` or `$'` will slow down **all** regex use within your program. Consult *perlvar* for `@-` to see equivalent expressions that won't cause slow down. See also *Devel::SawAmpersand*. Starting with Perl 5.10, you can also use the equivalent variables `${^PREMATCH}`, `${^MATCH}` and `${^POSTMATCH}`, but for them to be defined, you have to specify the `/p` (preserve) modifier on your regular expression.

```
$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
%+    Named capture buffers
%-    Named capture buffers, as array refs
```

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 a 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. Updated by the Perl 5 Porters.

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.

- *perlrebackslash* for a reference on backslash sequences.
- *perlrecharclass* for a reference on character classes.
- The *re* module to alter behaviour and aid debugging.
- *"Debugging regular expressions" in perldebug*
- *perluniintro*, *perlunicode*, *charnames* and *perllocale* 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.