Regular expressions in Perl

This document presents a tabular **summary** of the regular expression (regexp) syntax in Perl, then illustrates it with a collection of annotated **examples**.

Metacharacters

char	meaning
^	beginning of string
$	end of string
.	any character except newline
*	match 0 or more times
+	match 1 or more times
?	match 0 or 1 times; *or*: shortest match
|	alternative
( )	grouping; “storing”
[ ]	set of characters
{ }	repetition modifier
\	quote or special

To present a metacharacter as a data character standing for itself, precede it with \ (e.g. \. matches the full stop character . only).

In the table above, the characters themselves, in the first column, are links to descriptions of characters in my *The ISO Latin 1 character repertoire - a description with usage notes*. Note that the physical appearance (glyph) of a character may vary from one device or program or font to another.

Repetition

*a**	zero or more *a*’s
a+	one or more *a*’s
a?	zero or one *a*’s (i.e., optional *a*)
a{*m*}	exactly *m* *a*’s
a{*m*,}	at least *m* *a*’s
a{*m*,*n*}	at least *m* but at most *n* *a*’s
repetition?	same as *repetition* but the *shortest* match is taken

Read the notation *a*’s as “occurrences of strings, each of which matches the pattern *a*”. Read *repetition* as any of the repetition expressions listed above it. Shortest match means that the shortest string matching the pattern is taken. The default is “greedy matching”, which finds the longest match. The *repetition*? construct was introduced in Perl version 5.

Special notations with \

Single characters

\t	tab
\n	newline
\r	return (CR)
\x*hh*	character with hex. code *hh*

“Zero-width assertions”

\b	“word” boundary
\B	not a “word” boundary

Matching

\w	matches any *single* character classified as a “word” character (alphanumeric or “_”)
\W	matches any non-“word” character
\s	matches any whitespace character (space, tab, newline)
\S	matches any non-whitespace character
\d	matches any digit character, equiv. to [0-9]
\D	matches any non-digit character

Character sets: specialities inside [...]

Different meanings apply inside a character set (“character class”) denoted by [...] so that, **instead of** the normal rules given here, the following apply:

[*characters*]	matches any of the characters in the sequence
[*x*-*y*]	matches any of the characters from *x* to *y* (inclusively) in the ASCII code
[\-]	matches the hyphen character “-”
[\n]	matches the newline; other single character denotations with \ apply normally, too
[^*something*]	matches any character *except* those that [*something*] denotes; that is, immediately after the leading “[”, the circumflex “^” means “not” applied to all of the rest

Examples

expression	**matches...**
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b’s followed by a c
ab{2,}c	an a followed by at least two b’s followed by a c
ab*c	an a followed by any number (zero or more) of b’s followed by a c
ab+c	an a followed by one or more b’s followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc

[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)
perl\B	perl when *not* followed by a word boundary (e.g. in perlert but not in perl stuff)

Examples of simple use in Perl statements

These examples use very simple regexps only. The intent is just to show *contexts* where regexps might be used, as well as the effect of some "flags" to matching and replacements. Note in particular that matching is by default *case-sensitive* (Abc does not match abc unless specified otherwise).

s/foo/bar/;
replaces the *first* occurrence of the exact character sequence foo in the "current string" (in special variable $_) by the character sequence bar; for example, foolish bigfoot would become barlish bigfoot

s/foo/bar/g;
replaces *any* occurrence of the exact character sequence foo in the "current string" by the character sequence bar; for example, foolish bigfoot would become barlish bigbart

s/foo/bar/gi;
replaces any occurrence of foo *case-insensitively* in the "current string" by the character sequence bar (e.g. Foo and FOO get replaced by bar too)

if(m/foo/)...
tests whether the current string contains the string foo

Date of creation: 2000-01-28. Last revision: 2007-04-16. Last modification: 2007-05-28.

Finnish translation – suomennos: Säännölliset lausekkeet Perlissä.

The inspiration for my writing this document was *Appendix : A Summary of Perl Regular Expressions* in Pankaj Kamthan's *CGI Security : Better Safe than Sorry*, and my own repeated failures to memorize the syntax.

This page belongs to section *Programming* of the free information site *IT and communication* by Jukka "Yucca" Korpela.