WIKIBOOKS

Regular Expressions/POSIX-Extended Regular Expressions

The more advanced "extended" regular expressions can sometimes be used with Unix utilities by including the command line flag "-E". Other Unix utilities, like awk, use it by default.

The main difference is that some backslashes are removed: `\{…\}` becomes `{…}` and `\(…\)` becomes `(…)`. Examples:

- "`[hc]+at`" matches with "`hat`", "`cat`", "`hhat`", "`chat`", "`hcat`", "`ccchat`" etc.
- "`[hc]?at`" matches "`hat`", "`cat`" and "`at`"
- "`([cC]at)|([dD]og)`" matches "`cat`", "`Cat`", "`dog`" and "`Dog`"

The characters `(`,`)`,`[`,`]`,`.`,`*`,`?`,`+`,`|`,`^` and `$` are special symbols and have to be escaped with a backslash symbol in order to be treated as literal characters. For example:

> "`a\.(\(|\))`" matches with the string "`a.)`" or "`a.(`"

Modern regular expression tools allow a quantifier to be specified as non-greedy, by putting a question mark after the quantifier: `(\[\[.*?\]\])`.

Contents

Table of metacharacters

The following metacharacters are used:

Metacharacter	Description
`.`	Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor, character encoding, and platform specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, `a.c` matches "*abc*", etc., but `[a.c]` matches only "*a*", ".", or "*c*".
`[ ]`	A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches "*a*", "*b*", or "*c*". `[a-z]` specifies a range which matches any lowercase letter from "*a*" to "*z*". These forms can be mixed: `[abcx-z]` matches "*a*", "*b*", "*c*", "*x*", "*y*", or "*z*", as does `[a-cx-z]`. The `-` character is treated as a literal character if it is the last or the first (after the `^`) character within the brackets: `[abc-]`, `[-abc]`. Note that backslash escapes are not allowed. The `]` character can be included in a bracket expression if it is the first (after the `^`) character: `[]abc]`.
`[^ ]`	Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than "*a*", "*b*", or "*c*". `[^a-z]` matches any single character that is not a lowercase letter from "*a*" to "*z*". As above, literal characters and ranges can be mixed.
`^`	Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
`$`	Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.
BRE: `\( \)` ERE: `( )`	Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, `\n`). A marked subexpression is also called a block or capturing group.
`\n`	Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is theoretically **irregular** and was not adopted in the POSIX ERE syntax. Some tools allow referencing more than nine capturing groups.
`*`	Matches the preceding element zero or more times. For example, `ab*c` matches "*ac*", "*abc*", "*abbbc*", etc. `[xyz]*` matches "", "*x*", "*y*", "*z*", "*zx*", "*zyx*", "*xyzzy*", and so on. `\(ab\)*` (in BRE) or `(ab)*` (in ERE) matches "", "*ab*", "*abab*", "*ababab*", and so on.
BRE: `\+` ERE: `+`	Matches the preceding element one or more times. For example, `ab\+c` (in BRE) or `ab+c` (in ERE) matches "*abc*", "*abbbc*", etc., but not "*ac*", `[xyz]\+` (in BRE) or `[xyz]+` (in ERE) matches "*x*", "*y*", "*z*", "*zx*", "*zyx*", "*xyzzy*", and so on. `\(ab\)\+` (in BRE) or `(ab)+` (in ERE) matches "*ab*", "*abab*", "*ababab*", and so on.
BRE: `\?` ERE: `?`	Matches the preceding element one or zero times. For example, `ab\?c` (in BRE) or `ab?c` (in ERE) matches either "*ac*" or "*abc*", while `\(ab\)\?` (in BRE) or `(ab)?` (in ERE) matches "" or "*ab*".
BRE: `\\|` ERE: `|`	Matches the preceding element or the following element. For example, `abc\\|def` (in BRE) or `abc|def` (in ERE) matches either "*abc*" or "*def*".
BRE: `\{m,n\}` ERE: `{m,n}`	Matches the preceding element at least *m* and not more than *n* times. For example, `a\{3,5\}` (in BRE) or `a{3,5}` (in ERE) matches only "*aaa*", "*aaaa*", and "*aaaaa*".
BRE: `\{m\}` ERE: `{m}`	Matches the preceding element exactly *m* times.

BRE: `\{m,\}` ERE: `{m,}`	Matches the preceding element at least *m* times.
BRE: `\{,n\}` ERE: `{,n}`	Matches the preceding element not more than *n* times. For example, `ba\{,2\}b` (in BRE) or `ba{,2}b` (in ERE) matches only "*bb*", "*bab*", and "*baab*".

Character classes

The POSIX standard defines some classes or categories of characters as shown in the following table:

POSIX class	similar to	meaning
`[:upper:]`	`[A-Z]`	uppercase letters
`[:lower:]`	`[a-z]`	lowercase letters
`[:alpha:]`	`[[:upper:][:lower:]]`	upper- and lowercase letters
`[:alnum:]`	`[[:alpha:][:digit:]]`	digits, upper- and lowercase letters
`[:digit:]`	`[0-9]`	digits
`[:xdigit:]`	`[0-9A-Fa-f]`	hexadecimal digits
`[:punct:]`	`[.,!?:…]`	punctuation
`[:blank:]`	`[ \t]`	space and TAB characters only
`[:space:]`	`[ \t\n\r\f\v]`	blank (whitespace) characters
`[:cntrl:]`		control characters
`[:graph:]`	`[^\t\n\r\f\v]`	printed characters
`[:print:]`	`[^ \t\n\r\f\v]`	printed characters and space

Links:

- W:Regular_expression#Character_classes

- Character Classes that are Always Supported (http://www.boost.org/doc/libs/1_44_0/libs/regex/doc/html/boost_regex/syntax/character_classes/std_char_clases.html) at boost.org

Use in Tools

Tools and languages that utilize this regular expression syntax include:

- AWK - uses a superset of the extended regular expression syntax

Links

- POSIX Basic Regular Expressions (http://www.regular-expressions.info/posix.html) at regular-expressions.info
- POSIX Extended Regular Expression Syntax (http://www.boost.org/doc/libs/1_71_0/libs/regex/doc/html/boost_regex/syntax/basic_extended.html) at boost.org