Regular Expression

Original Notes by Song Guo
Converted to Presentation and Update by
Bing Li
What Are Regular Expressions?

- A regular expression is a pattern describing a certain amount of text.
- A “match” is the piece of text, or sequence of bytes or characters that pattern was found to correspond to by the regex processing software.
Regular Expression Engines

- A regular expression “engine” is a piece of software that can process regular expressions, trying to match the pattern to the given string.
  - POSIX, Perl, Vim……
  - The engine is part of a larger application and you do not access the engine directly.
  - Different regular expression engines are not fully compatible with each other.
  - It is not possible to describe every kind of engine and regular expression syntax (or “flavor”).
Literal Characters

• The most basic regular expression consists of a single literal character.

• e.g.: “Jack is a boy”, “a” matches the “a” after the “J”

• It can match the second “a” when you tell the regex engine to start searching through the string after the first match.

• Case sensitive. “cat” will match “cat” in “About cats and dogs”, but not “Cat” in “About Cats and Dogs”.
Special Characters

• Meta-characters - characters with special meanings. e.g.: [,],\,^,$,..

• A backslash cause the metacharacters to be treated as a literal character. It can also treat some literal characters to meta-characters.
  
  • Use “1\+1=2” to match “1+1=2” because “+” has a special meaning.
  
  • In vim, “+” is a literal character.

• Non-printable characters: e.g. \n for line feed.
  
  • whitespace : \t\r\n\v\f
Character Classes / Sets

• Character classes (Character sets) “[…]”: matches a single character contained within the brackets.
  
  • e.g.: “gr[ae]y” matches both “gray” and “grey”

• Negated Character Classes “[^…]”: matches a single character not contained within the brackets.
  
  • e.g.: “g[^u]” matches “a q followed by any single character but not u”
Character Classes / Sets

• Shorthand Character Classes: [A-Z],[a-z],[0-9]

• “-” character is treated as a literal character if it is the first or the last character within the brackets. [-ab],[ab-]

• “]” character can be included in brackets if it is the first character.[]123

• Negated Shorthand Classes: [^a-z]
Character Classes / Sets

- \w: [A-Za-z0-9_]     \W: [^\w]
- \d: [0-9]             \D: [^\d]
- \s (vim: \_s): [ \t\r\n\v\f]  \S: [^\s]
Character Classes / Sets

• vim:
  • “\l”: [a-z] Lowercase letters
  • “\u”: [A-Z] Uppercase letters
  • “\s”: [ \t] space and tab
  • “\x”: [A-Fa-f0-9] Hexadecimal digits
  • “\p”: [\x20-\x7E] Visible and space characters
Quantifier

• Repeating Character Classes:
  
  • “?” (vim: “\?”): matches the preceding element zero or one time.
    
    • “ab?c” matches “ac” and “abc”
  
  • “+” (vim: “\+”): matches at least one time.
    
    • “ab+c” matched “abc”,”abbc”…but not “ac”
  
  • “*” : matches zero or more times
    
    • “ab*c” matched “ac”."abc”,”abbc”…”

Quantifier

- “{min,max}” (vim“\{…\}”): matches between min and max times.
  - “a{3,5}”: matched “aaa”, ”aaaa”, ”aaaaa”
  - “{0,}”: is the same as “*”
  - “{1,}”: is the same as “+”
  - “{#}” (vim“\{…\}”): matched exact # times
- “[1-9][0-9]{3}” matches a number between 1000 and 9999
Greediness

• Regex will match characters as much as possible.

• “<.+>” matches “<EM>first</EM>” but not only “<EM>”

• “?”, “*”, “+”, are greedy

• Make greedy become lazy.

  • Put a question mark behind the plus “<.+?>”
Greediness

- vim

- \{-\} : *? matches 0 or more of the preceding characters, as few as possible.

- \{-n,m\} : matches from n to m of the preceding characters, as few as possible.

- \{-n,\} : matches at least n of the preceding characters, as few as possible.

- \{-,m\} : matches at most m of the preceding characters, as few as possible.
Alternation with Vertical Bar

- “|” (vim: “\|”): matches either the pattern before | or after |
- “abc|def” matches “abc” and “def”
- “abc|def|ghi” matches “abc”, “def” and “ghi”
- “.”: matches all single character (except new line?)
Anchors

- “^”: matches the position before the first character in the string (or each line?)
- “$”: matches the position after the last character in the string (or each line?)
- “^\s+”: matches leading white spaces
- “\s+$”: matches trailing white spaces
- “^\d+$” does not match the “5” in “adsadsa5asdas”, but “\d+” does.
Word Boundaries

• “\b” is an anchor like the “^” and “$”. It matches at a position that is called a “word boundary”. This match is zero-length.

• Three positions that qualify as word boundaries
  • Before the first character, it the first character is a word character.
  • After the last character, if the last character is a word character.
  • Between two characters, where one is word character but the other one is not.
Word Boundaries

• "\bword\b" matches "word" only, not "aword" or "words"...

• "\B" is the negated version of "\b".

• Vim uses "\<" and "\>" to match the word boundaries.

• "\<word\>" matches "word" in vim
Sub-pattern

• “()” (in Vim, “\(\)”)  
  • The sub-pattern in parentheses is a whole part when it is quantified.  
  • e.g. “a(bc)*” matches “a”, “abc”, “abcabc”. 