Regular expressions in depth

Datto Engineering

Presented By: Fred Mora – July 2014

Agenda

- What are regexes
- What should you care
- Origins
- Syntax
- Greed is good
- Grouping and matching
- PHP and regexes

What are regular expressions?

- Regexes are a formal notation for representing set of strings
  - A regex represents a machine or program that can emit certain strings
  - Such program is equivalent to a matcher for these same strings
- Regexes are:
  - a human-readable notation
  - a specification for string matching in many computer languages.
- Regexes are used to match and select text in files or string.

Why should you care

- Text is the main communication means between machines and humans
  - A browser in a GUI shows fancy text, but it's still text.
  - Text is the best way to express complex ideas.
- Due to HTML and XML prevalence, text is also prevalent between machines.
  - Binary interfaces are limited to IPC within the same cluster
  - Bandwidth is rarely so critical that you have to exchange only binary buffers.
- So you will spend a lot of time handling text

The origins

- The first Unix machines came with text processors
- Early regex libraries shipped with Unix
- Basic Unix tools like sed, awk, grep use regexes
- Regexes are heavily used in Perl, and thus in PHP

Basic regex expressions

Regex	*Comments*
/abc/	A normal string is a regex that matches itself. Not too useful.
/[a-c]/	Character classes are defined between brackets. A range is a class that matches all characters between the boundaries. Here, matches a, b or c.
/[1-9A-Z]/ **/[XYZT]/** **/[^0-9a-fA-F]/**	Compound range. Matches any char between 1 to 9 or A to Z Matches X, Y, Z or T Matches anything but what's in the brackets (here, hex digits)
\s/and **\S**	**\s** matches a white space. Matches a space, but also a tab, \r, \n **\S** is the opposite, it matches a non-white space char
\d and **\D**	**\d** is the same as **[0-9]**. **\D** matches any non-digit char.
\w and **\W**	**\w** matches any "word" char as defined in Perl, that is, letter, digits and underscore. **\W** is the opposite.
[:alnum:], **[xdigit:]**, **[:punct:]**	POSIX character classes. Respectively match alphanumerical chars, hexadecimal digits, punctuation marks. There are others, e.g., **[:upper:]** for **[A-Z]**.
/abc|def/	The pipe symbol means "pick one". Example: /**pet = cat|dog/** matches "pet = cat" and "pet = dog"

Quantifiers

Scribble	*Meaning*	*Example*
.	Dot - Any char	**/abc.e/** matches “abcde”, “abcXe”. A literal dot is escaped with backslash, as in **\.**
*	Zero or more time	**/ab*c/** matches “ac”, “abc”, “abbbbbbbc”
?	Zero or one time	**/ab?c/** matches “ac”, “abc”, but not “abbc”
+	One of more times	**/ab+c/** matches “abc”, “abbc”, “abbbbbbbc”
{n,m} **{n,}** **(n)**	N to m times N or more times Exactly n times	**/ab{2,4}c/** matches “abbbc” **/ab{2,}c/** matches “abbbc” **/ab{2}c/** matches “abbc”

Anchors

Scribble	*Meaning*	*Example*
^	Start of string	**/^R/** matches “Rapid”, “RegEx” **/^[A-C]olt/** matches “Bolt” and “Colt” but not “Dolt”
$	End of string	**/[A-Z]\d{2}$/** matches “XB70” or “B52”, but not “F104” or “B1B”

Greed is good

- By default, quantifiers are greedy. That is, they match as many chars as they can. Example:
  /<a href=".*<\/a>/ matches the red portion of this HTML text:

  Some links:
  Our company
 Our technology
  Happy reading.

- In this case, the greedy match is probably not what was intended, The star quantifier is greedy. Make it lazy with **?** as follows:
  /<a href=".*?<\/a>/ matches the red portion of this HTML text:

  Some links:
  Our company
 Our technology
  Happy reading.

Grouping and matching

- Groups are denoted by parentheses.
- Within the regex, the previously *i*-th matched group is denoted by *i* (\1, \2, etc.)
- Examples:

Regex	***Matches***	***Groups***
/(\w+)\s+\1/	“blah blah” “foo foo” But not “foo bar”	'blah' 'foo'
/file:\s+(.+?), size:\s+(\d+)\s*([KMG]?)B/i	“file: foo.txt, size: 123 kB” “File: bar, size: 45Gb” “file: qux, size: 789 B”	'foo.txt', '123', 'k' 'bar', '45', 'G' 'qux', '789', ''

- Note the i modifier for case insensitivity

Commenting regexes

- Even relatively simple regexes quickly start looking like a cat has jumped on the keyboard.
- If your code will be maintained by less enlightened coders, comment your regexes with the x modifier.
- Example:
  - Before:
    /file:\s+(.+?), size:\s+(\d+)\s*([KMG]?)B/i
  - After:

```
/file:      # Literal
 \s+        # One or more spaces
 (.+?),     # Group: All chars up to a comma
 \s+        # One of more spaces
 size:      #Literal
 \s+        # One of more spaces
 (\d+)      # Group: One or more digits
  \s*       # Optional space
  ([KMG]?)  # Group: Optional multiplier
 B
/xi
```

Coding examples in PHP

```
$regex = "/file: # Literal
    \s+          # One or more spaces
    (.+?),       # Group: All chars up to a comma
    \s+          # One of more spaces
    size:        #Literal
    \s+          # One of more spaces
    (\d+)        # Group: One or more digits
    \s*          # Optional space
    ([KMG]?)     # Group: Optional multiplier
    B
/xi";  // x = commented regex, i = case-insensitive
$str = "file: bar, Size: 789 Kb\n";

preg_match($regex, $str, $matches);
print print_r($matches, true);

Run:
Array
(
    [0] => file: bar, Size: 789 Kb
    [1] => bar
    [2] => 789
    [3] => K
)
```

Coding examples in PHP – cont'd

```
$marker = 'Error SQL123';
$regex = "/$marker: Application (\w+) cannot access table (\w+)/";

$loglines = file('log.txt');  // Read whole file in memory. NOT PRODUCTION CODE!
foreach ($loglines as $line) {
  if (preg_match($regex, $line, $matches) == 0) {
    continue;
  }
  print "Problem with app ${matches[1]}, table ${matches[2]}\n";
 // More processing
}
------------------------------
File log.txt:

2014-07-09 10:08:59 some line
2014-07-09 10:09:03 nope, still no matche
2014-07-09 10:09:11 Hey, look: Error SQL123: Application foobar cannot access table QUX, oh noes
2014-07-09 10:10:01 Not that one
2014-07-09 10:10:08 Another match: Error SQL123: Application baz cannot access table BLAH.
2014-07-09 10:10:40 And so on
------------------------------
Run:

Problem with app foobar, table QUX
Problem with app baz, table BLAH
```

References

- Some useful links:
  - Perl regex tutorial: http://perldoc.perl.org/perlretut.html
  - Online regex testers and debuggers:
    - http://regex101.com/
    - http://www.regexr.com/
- Don't overdo regexes. Some formats are too complex and need a full-blown parser.
- About overly complex regexes:
  “Some people, when confronted with a problem, think “I know, I'll use regular expressions.” Now they have two problems.” – Jamie Zawinski <jwz@netscape.com>, 12 Aug 1997, alt.religion.emacs

Questions?