Regular Expressions in Perl

An Introduction.

By

Mike Burns
mburns.lungching@gmail.com

Liquidity Services Inc.

Regexes in Perl

- Comparison:
  ```
  if ($string =~ /foo/) { ... }
  ```
- Selection:
  ```
  $string =~ /foo(.*)bar/;
  my $between_foo_bar = $1;
  ```
- Replacement:
  ```
  $string =~ s/foo/bar/;
  ```

String Comparison

Find the literal characters “red”.

```
my $string = ‘red riding hood’;

if ($string =~ /red/) {
    print “$string has 'red' in
it!\n”;
}

if ($string = =~ /din/) {
   print “$string has 'din' in
  it!\n”;
}
```

/red/

- bar**red**
- **red**is
- ti**red**
- caught **red** handed
- **red**, purple, and blue shirt

Character Classes

- \w Match a "word" character (alphanumeric plus "_", plus other connector punctuation chars plus Unicode marks).
- \W Match a non-“word” character.
- \s Match a whitespace character.
- \S Match a non-whitespace character.
- \d Match a decimal digit character.
- \D Match a non-digit character.
- ...

General Character Classes

- digits => [0-9]
- vowels => [aeiou]
- letters => [A-Za-z]

The \d assertion.
Match digits.

/\d/

- barred
- blah
- 12345
- There are 12 things.
- Foo1 is my username.

```
my $string = 'Foo1 is my username.';

if ($string =~ /\d/) {
    print "$string has a number in it!\n";
}
```

Metacharacters

- \ Quote the next metacharacter.
- ^ Match the beginning of the line.
- . Match any character (except newline).
- $ Match the end of the line.
- | Alternation.
- ( ) Grouping.
- [ ] Bracketed character class.

The ^ metacharacter.

Match the beginning of the line.

/^red/

- barred
- redis
- tired
- caught red handed
- **red**, purple, and blue shirt

```
if ($string =~ /^red/) {
   print "$string begins in ‘red’!\n";
}
if ($string =~ /^riding/) {
    print "$string begins in ‘riding’!\n";
}
```

Quantifiers

- * Match 0 or more times
- + Match 1 or more times
- ? Match 1 or 0 times
- {n} Match exactly n times
- {n,} Match at least n times
- {n,m} Match at least n but not more than m times

String Selection

The title out of an HTML document.

Conceptually

- Find the literal characters “<title>”.
- Followed by zero or more characters which will be captured and returned.
- Then ends with the literal characters “</title>”.

What We’ll Need

- ( ) To capture and return the title.
- . To match any character (except newline).
- * To match zero or more times.
- \ To escape what would normally not be a literal character.

/<title>/

```
<html>    <head>
<title>Example.com</title>
 </head>    <body>
Hello world!
</body></html>
```

/<title>./

```
<html>    <head>
<title>Example.com</title>
 </head>    <body>
Hello world!
</body></html>
```

/<title>.*/

```
<html>    <head>
<title>Example.com</title>
 </head>    <body>
Hello world!
</body></html>
```

```
/<title>(.*)/
```

```
<html>   <head>
<title>Example.com</title>
 </head>   <body>
Hello world!
</body></html>
```

```
/<title>(.*)<\/title>/
```

```
<html>    <head>
<title>Example.com</title>
 </head>    <body>
Hello world!
</body></html>
```

/<title>(.*)</title>/

```
$html =~ /<title>(.*)<\/title>/;
print “The title is $1\n”;
```

Date Capture Example

```
my ($year, $month, $day, $time) =
$string =~ /(\d{4})-(\d{2})-(\d{2})
(\d{2}:\d{2})/;



print "year: $year month: $month day:
$day time: $time\n";
```

Modifiers

- m Treat string as multiple lines. Affects “^” and “$”.
- s Treat string as single line. Affects “.”.
- i Case-insensitive pattern matching.
- x Allow whitespace and comments.
- g Global matching (match more than once).
- ...

String Replacement

Remove county name suffixes.

Remove county name suffixes.

my $string = ‘Los Angeles CountyHarris CountyLafayette Parish’;

Conceptually

- For each line.
- Match for a particular set of strings at the end of the line.
- Replace the match with an empty string.

What We’ll Need

- $ Match the end of the line.
- | Alternation metacharacter.
- ( ) Group alternating patterns.
- m Treat string as multiple lines.
- g Modifier to do global matching.

```
/(County|Parish)/
```

Los Angeles
CountyHarris
CountyLafayette
Parish

```
/(County|Parish)/g
```

Los Angeles
CountyHarris
CountyLafayette
Parish

```
$string =~ s/ (County|Parish)$//g
```

$string eq ‘Los Angeles Harris Lafayette’;

Matching an IP address.

0.0.0.0

to

255.255.255.255

What We’ll Need

- [ ] Bracketed character class to match digits.
- ( ) Grouping metacharacter.
- | Alternation metacharacter.
- \ To escape what would normally not be a literal character.

Matching a number from 0 to 255.

```
25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9]
```

```
/^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|
[1-9][0-9]|[0-9])\.(25[0-5]|2[0-4][0-
9]|1[0-9][0-9]|[1-9][0-9]|[0-9])\.
(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-
9][0-9]|[0-9])\.(25[0-5]|2[0-4][0-
9]|1[0-9][0-9]|[1-9][0-9]|[0-9])$/
```

Now do it four times with periods between.

```
/^(25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)){3}$/
```

This can be simplified.

```
my $re = qr/25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d/;
if ($ip =~ /^($re)\.($re)\.($re)\.($re)$/) {
    ...
}
```

There is still something wrong.

```
25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9]
```

This is brittle and overly complex.

```
use List::MoreUtils qw( all );

my @parts = split(/\./, $ip);

if (
    @parts == 4 and
    all { int($_)==$_ and $_>=0 and $_<=255 }
@parts
) {
    ...
}
```

Sometimes there are better solutions.

https://metacpan.org/module/Regexp::Common::net

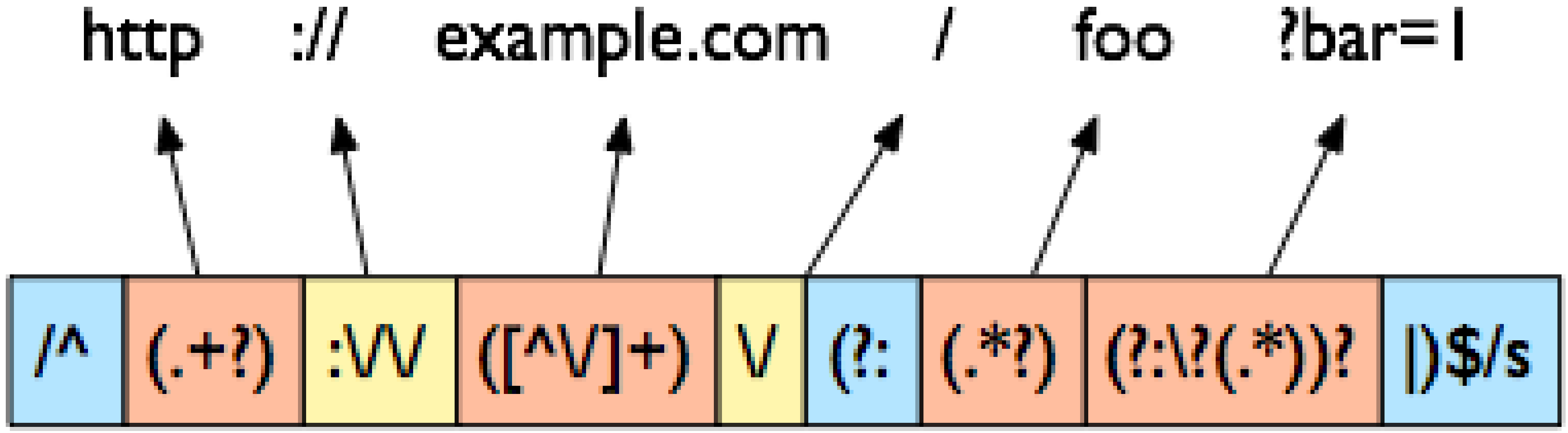
```
/^(.+?):\/\/([^\/]+)\/(?:(.*?)(?:\?(.*))?|)$/s
```

/^(.+?):\

KABOOM!

(.*?)(?:\?

/^(.+?):\/\/([^\/]+)\/(?:(.*?)(?:\?(.*))?|)$/s



A loose URI pattern.

Comment

```
/^(.+?)              # http
    :\/\/           # ://
    ([^\/]+)         # example.com
    \/              # /
    (?:
        (.*?)        # foo
        (?:
            \?(.*)   # ?bar=|
        )?|
    )$
/sx
```

The End
Questions?

Original presentation by

bluefeet@gmail.com
Aran Deltac
bluefeet@gmail.com