Regular Expressions in Perl

An Introduction.

Regexes in Perl

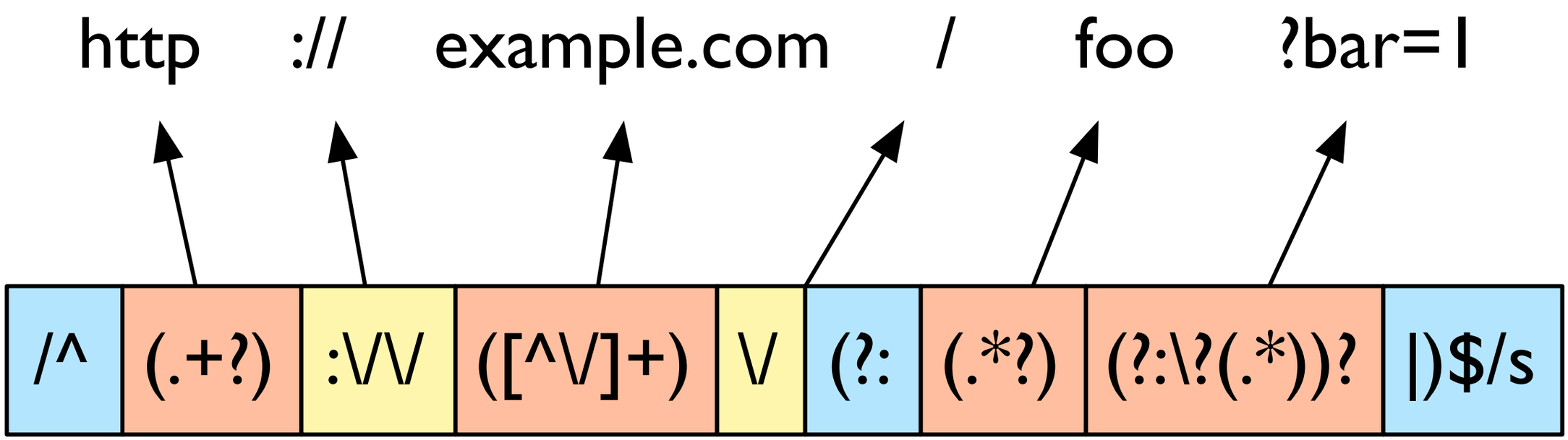
- Comparison:
```
if ($string =~ /foo/) { ... }
```
- Selection:
```
$string =~ /foo(.*)bar/;
my $between_foo_bar = $1;
```
- Replacement:
```
$string =~ s/foo/bar/;
```

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

/^(.+?):\/\/(

?(.*))?|)$/s

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



A loose URI pattern.

String Comparison

Find the literal characters “red”.

/red/

mZredZ

m{red}

```
my $string = ‘red riding hood’;

if ($string =~ /red/) {
    print “$string has ‘red’ in it!’;
}
```

/red/

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

The \b assertion.

Match a word boundary.

/\bred\b/

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

The ^ metacharacter.

Match the beginning of the line.

/^red\b/

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

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>
```

m{<title>(.*)</title>}

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

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

String Replacement

Remove county name suffixes.

```
my $string =
'Los Angeles County
Harris County
Lafayette 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+County

Harris+County

Lafayette+Parish

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

Los+Angeles+County

Harris+County

Lafayette+Parish

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

Los+Angeles+County

Harris+County

Lafayette+Parish

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

Los+Angeles+County

Harris+County

Lafayette+Parish

```
/ (County|Parish)$/gm
```

Los+Angeles+County

Harris+County

Lafayette+Parish

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

```
$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]
```

Now do it four times with periods between.

```
/^(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])$/
```

This can be simplified.

```
/^(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}$/
```

And, simplified even more.

```
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.

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

Sometimes there are better solutions.

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

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

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

The End

Questions?

bluefeet@gmail.com

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.

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).
- ...

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

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.
- ...

Assertions

- \b Match a word boundary.
- \B Match except at a word boundary.
- \A Match only at beginning of string.
- \Z Match only at end of string, or before newline at the end.
- \z Match only at end of string.
- ...