
http://xkcd.com/208/

# Regular Expressions for Fun and Profit

## or spinning, for your cpu

Brad Lhotsky
brad.lhotsky@gmail.com
@reyjrar

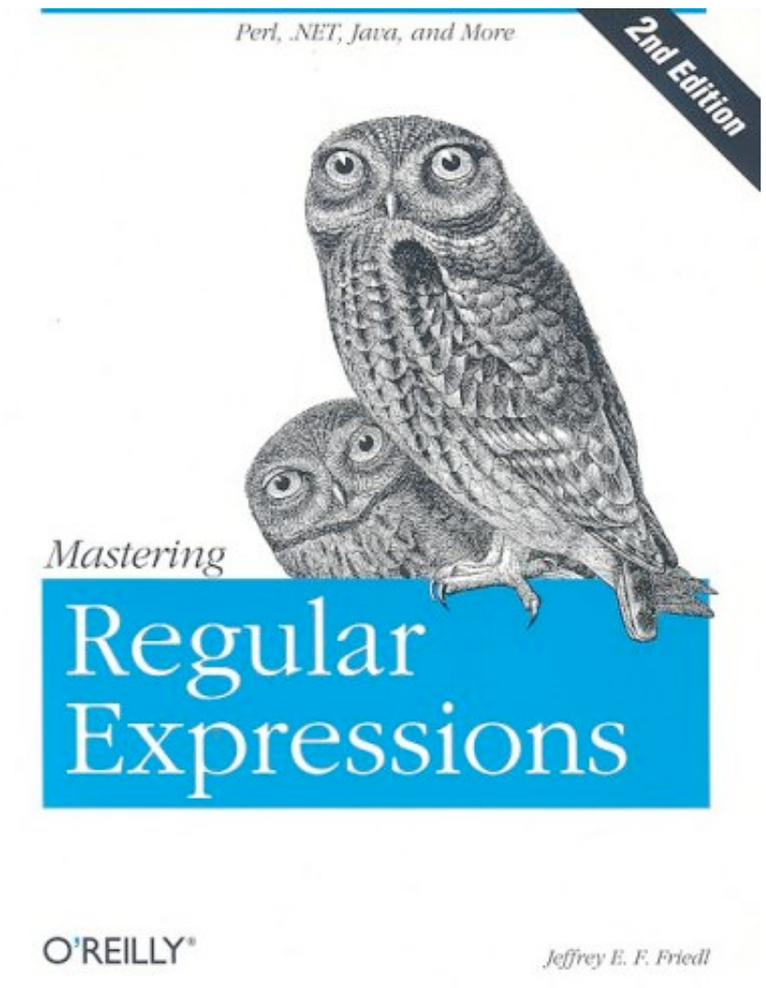
# Disclosure and Disclaimer

- Beginner level
- Contains trace amounts of mercury
- May cause internal bleeding
- Boring Tables Ahead
- Slides Perl based

# RTFM

- Perl:
  - perldoc perlre
- Python:
  - pydoc re
- Ruby:
  - class is Regexp, you figure out how to pull up docs ;)
- PHP:
  - PCRE is (preg_*)
- libpcre
  - man pcre
  - man pcrepattern

# ~~I~~rregular Expressions




- Pattern Matching
- *Very* Powerful
- Extremely Steep Learning Curve
- Did my modem just throw up into my code?
- “regex”, “regexp”, “regex engine”

Example: 0 - 255, with or without leading zeros

```
(25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})
```

# Basics of Regular Expressions

- Operate on strings, not numbers
- Character by character basis
- Will try every possible variation to match your string
- Wants to make you happy
- KNOW YOUR DATA!!

# Regex : Modifiers

| Modifier | Description |
|---|---|
| i | Case insensitive matching |
| g | Globally match (don't stop at the first match) |
| m | Treat multi-line strings as a single string |
| x | Extend readability by allowing whitespace and comments |

```
$string =~ m/^a/i;
```

# Regex: Grouping

| Symbol | Description | Notes |
|---|---|---|
| `[a-z]` | Character Class | Expands to:<br>abcdefghijklmnopqrstuvwxyz |
| `[^a-z]` | Inverted Character Class | Matches characters except those specified |
| `(...)` | Grouping w/ Capture | Stores the matched substring in `$1`, `$2`, … |
| `(?:...)` | Grouping w/o Capture | Allows a programmer to group without capturing |

# Regex: Special Classes

| Symbol | Meaning | Opposite |
|---|---|---|
| `\w` | Matches any word character [a-zA-Z0-9_] (includes utf8 if applicable) | `\W` |
| `\d` | Matches any digit [0-9] (includes utf8 if applicable) | `\D` |
| `\s` | Matches all whitespace characters (includes utf8 if applicable) | `\S` |

# Regex Meta-Characters

| Symbol | Description |
|---|---|
| \ | Escapes the next character |
| . | Matches any *single* character |
| a \| z | Matches *a* **or** *z* |

# Regex : Anchors

| Symbol | Description |
|---|---|
| ^, \A | Matches the beginning of the string |
| $, \Z | Matches the end of the string |
| \b | Matches at a word boudary, between word and non-word characters |

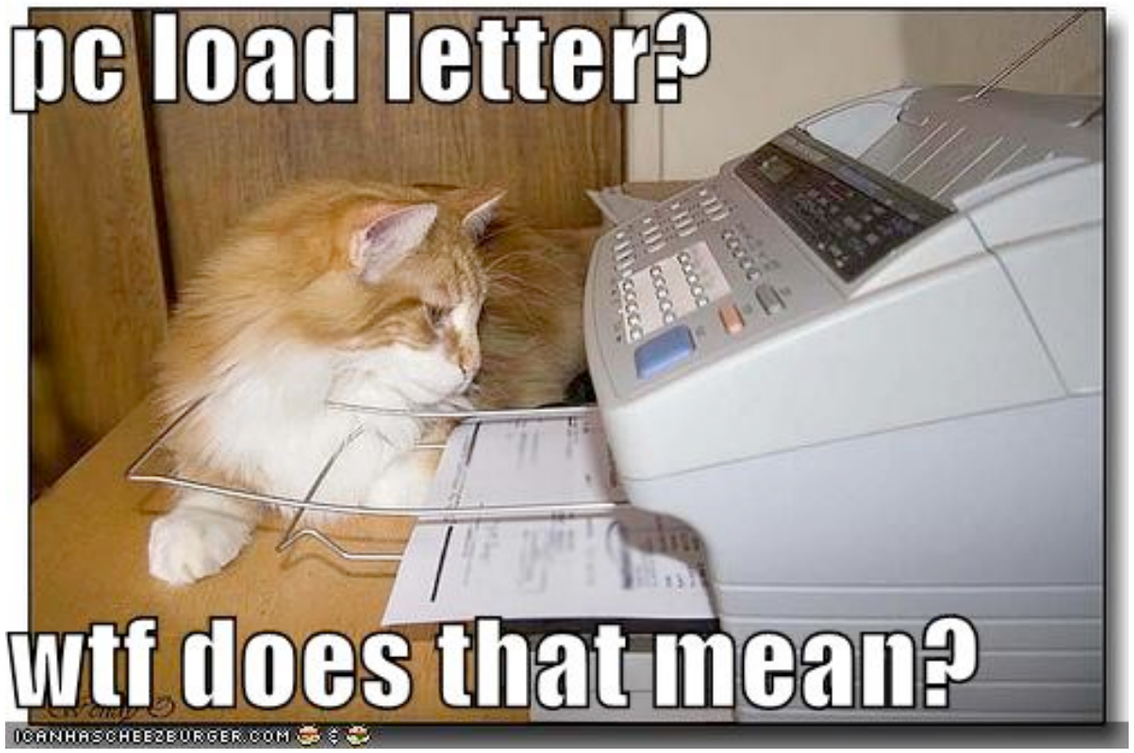
# Regex: Quantifiers

| Quantifier | Meaning |
|---|---|
| * | Matches if found *0 or more times* |
| + | Matches if found *1 or more times* |
| ? | Matches if found *0 or 1 times* |
| {x,y} | Matches if found *between* x *and* y *times* |
| {x,} | Matches if found *at least* x *times* |
| {,y} | Matches if found *no more than* y *times* |
| {x} | Matches if found *exactly* x *times* |

*** *These are all* greedy *quantifiers* ***

# Regular Expression Syntax

```
# Common usage:
$string =~ /abc/;

# Expands to:
$string =~ m/abc/;

# Case insensitive
$string =~ /abc/i;

# Single Substitution:
$string =~ s/abc/def/;

# Global & Insensitive:
$string =~ s/abc/def/gi;
```



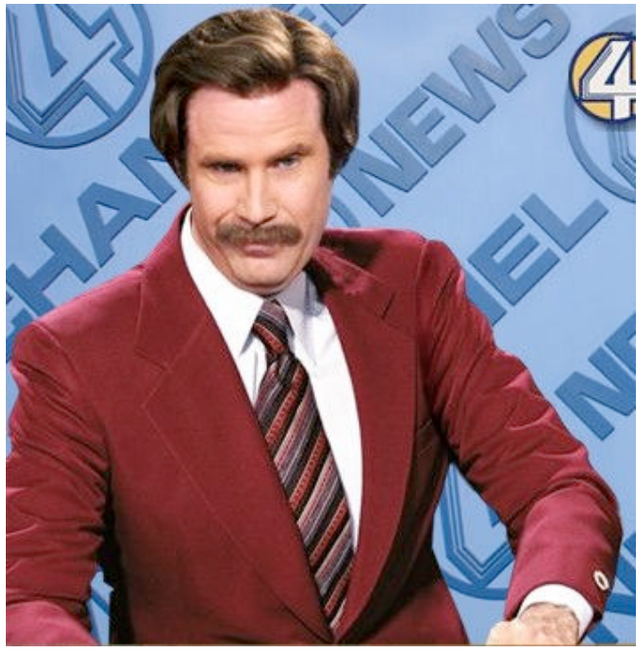
# Reading regex like the engine

```
# Simple Example:
my $string = ‘abc’;
$string =~ m/abc/;
# READ AS:
# ‘a’ followed by
# ‘b’ followed by
# ‘c’
```

*"Big mouthfuls often choke." (Italian Proverb)*

```
# Greed and you
my $string = 'abcdefgh';
$string =~ m/.*abc/;
# READ AS:
# .* takes 'abcdefgh' = Match Fails
# .* gives back 'h' = Match Fails
# .* gives back 'g' = Match Fails
# ...
# After .* gives back 'a'
# Engine checks,
# 'a' => SUCCESS
# followed by 'b' => SUCCESS
# followed by 'c' => SUCCESS
# MATCH SUCCESS
```

# Simple Examples: Character Classes




```
my $string = '002 Ron Burgundy - Stay Classy';

# Check if $string starts with a Number
my $test1 = $string =~ /^[0-9]/;  # 1

# Check for starts with 1 or more numbers
my $test2 = $string =~ /^[0-9]+/; # 1

# Check for 3 numbers
my $test3 = $string =~ /^[0-9]{3}/; # 1
```

# Simple Examples:
## Inverted Character Classes


```
my $string = ‘003 Ron Burgundy - Go Fuck
Yourself’;
```

```
# Check if $string starts with a Number
my $test1 = $string =~ /^[^0-9]/;  # 0

# Contains “Bad data” ? (input sanitation)
my $test2 = $string =~ /[^a-zA-Z0-9 \-]/; # 0
```

# Gotcha: Our first try to match an IP

```
my $ip = '127.0.0.1';

# Is it an IP
my $isip = $string =~ /[1-255](\.[0-255]){3}/;
# $isip = 0
```

**QDB** **Quote #38640**

Paypal Donate

Home / Latest / Browse / Random >0 / Top 100-200 / **Add Quote** / ModApp / Search / #

**#38640** +(956)- [X]

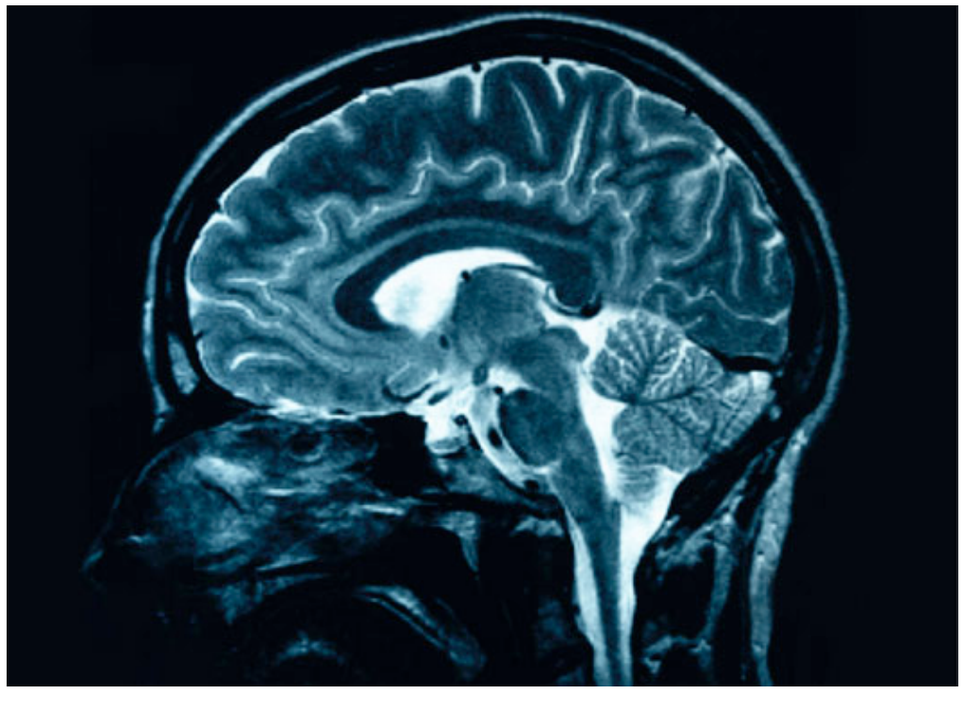
```
<Myren> someone ping flood this bastard please: 127.0.0.1
<darthv> ok
*** darthv has quit IRC (Ping timeout)
```

Home / Latest / Browse / Random >0 / Top 100-200 / **Add Quote** / Search / ModApp

0.0016 20774 quotes approved; 6106 quotes pending

Hosted by Idologic: high quality reseller and dedicated hosting.

# Gotcha: Regex Compilation



```
my $ip = '127.0.0.1';

# Is it an IP
my $isip = $string =~ /[1-255](\.[0-255]){3}/;
# Compiles as:
# /[125](\.[0125]){3}

# Regex operate on CHARACTERS
```

# Incorrectly matching an IP

```
my $ip = '127.0.0.1';

# Is it an IP
my $isip = $string =~ /\d+\.\d+\.\d+\.\d+/;
# $isip = 1
# Also matches 888.888.888.888
# or 8.888888888.8888888888.8
/\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/;
# Still matches 888.888.888.888
# We can shorten it:
/\d{1,3}(\.\d{1,3}){3}/;
# Might be "good enough"
```

# Is ‘good enough’ good enough?

```
#!/usr/bin/env perl

use strict;
use warnings;

use Regexp::Common qw(net);
use Benchmark qw(cmpthese);

my $string = "sdglkshdglhsdlghsdlkhgslkdg 10.10.50.35 asfasfasgagsdgsdg";

cmpthese( 1_000_000, {
    good_enough => sub { $string =~ /[0-9]{1,3}(\.[0-9]{1,3}){3}/; },
    properly_done => sub { $string =~ /$RE{net}{IPv4}/; },
});
```

# Results

```
$ perl goodenough.pl
                  Rate properly_done   good_enough
properly_done  11273/s             --          -98%
good_enough   581395/s          5058%            --


$ perl --version

This is perl 5, version 12, subversion 2 (v5.12.2)
built for i686-linux

Copyright 1987-2010, Larry Wall
```

# on greed and laziness…

- Greedy quantifiers are ambitious and consume as much as they can to allow the ENTIRE regex to match


- Non-greedy Quantifiers are lazy and consume only enough of the string that is necessary to allow the ENTIRE regex to match

# Regex: Lazy Quantifiers

| Quantifier | Meaning |
|---|---|
| `*?` | Matches if *0, or more times if needed* |
| `+?` | Matches if *1, or more times if needed* |
| `{x,y}?` | Matches if *x times, up to* y *if needed* |
| `{x,}?` | Matches if x *times, more if needed* |
| `{,y}?` | Matches if *0 times, up to y times if needed* |

# Greedy vs Lazy

```
my $string = '1 2 3 4 5 6 7 8 9';

$string =~ /^.*([0-9]).*/;
# $1 captures '9'

$string =~ /^.*?([0-9]).*?/;
# $1 captures '1'
```

# Alternatives (Perl)

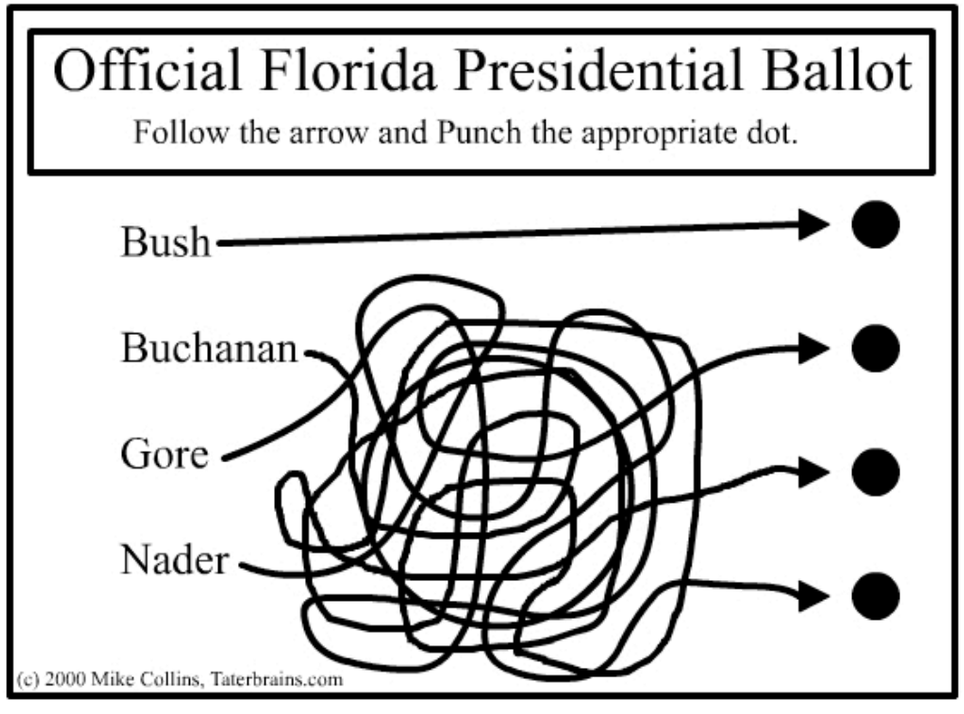
- `perldoc -f substr`
- `perldoc -f index`
- `perldoc -f unpack`
- http://search.cpan.org/dist/Regexp-Common/

# Testing some ideas..

```
#!/usr/bin/env perl

use strict;
use warnings;

use Benchmark qw(:all);

my $STRING = "Oct  5 18:05:31 fierydeath kernel: DefaultReject IN=eth0 OUT=
MAC=ff:ff:ff:ff:ff:ff:00:10:dc:ca:26:c0:08:00 SRC=1.3.4.5 DST=1.2.3.255 LEN=244
TOS=0x00 PREC=0x00 TTL=128 ID=32002 PROTO=UDP SPT=138 DPT=138 LEN=224";
```

```
my $regex = '(\w+\s+\d+\s+\d+\:\d+:\d+)';

cmpthese( 1_000_000, {
    greedy => sub { my ($date) = ($STRING =~ /^.*$regex.*$/); },
    lazy => sub { my ($date) = ($STRING =~ /^.*?$regex.*?$/); },
    regex => sub { my ($date) = ($STRING =~ /^$regex/); },
    use_substr => sub { my ($date) = substr($STRING, 0, 15 ); },
    check_index => sub { index( 'LEN=224', $STRING ); },
    check_regex => sub { $STRING =~ /LEN=224/; },
});
```

# Understand your data, Understand your options

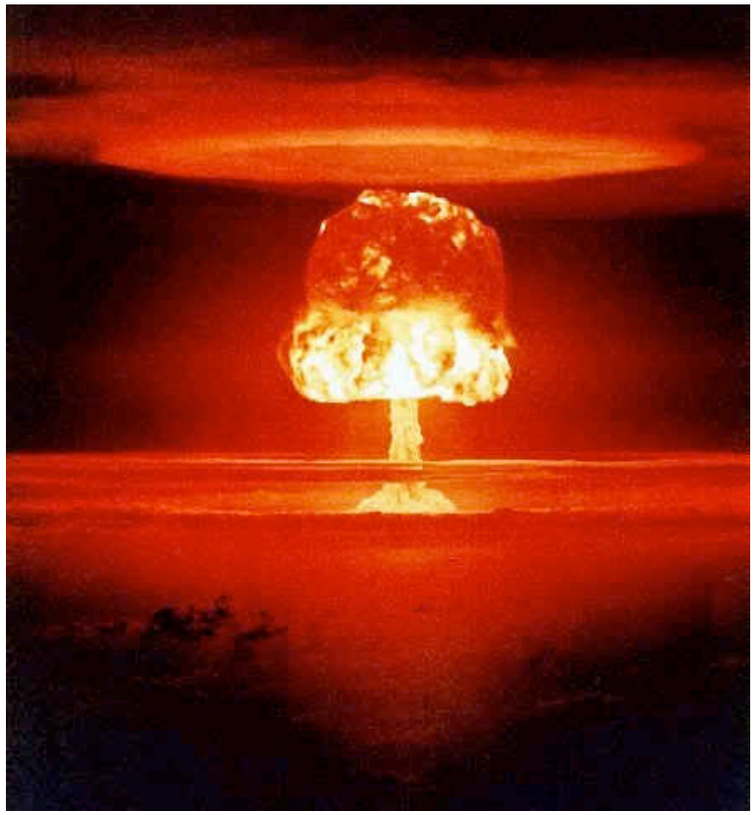



```
              Rate  greedy    lazy   regex use_substr check_regex check_index
greedy       13935/s      --   -78%   -96%       -99%       -99%       -100%
lazy         63776/s    358%     --   -83%       -97%       -98%        -99%
regex       370370/s   2558%   481%     --       -83%       -86%        -95%
use_substr 2127660/s  15168%  3236%   474%         --       -21%        -72%
check_regex 2702703/s 19295%  4138%   630%        27%         --        -65%
check_index 7692308/s 55100% 11962%  1977%       262%       185%          --

$ perl --version

This is perl 5, version 12, subversion 2 (v5.12.2) built for i686-linux

Copyright 1987-2010, Larry Wall
```

# The Right Tool for the Job

The Job

SOLUTION:



# Cool Tool: txt2regex

```
[.]quit  [0]reset  [*]color                                    ^txt2regex$
[|]or  [(]open group
                                                          !! not supported
 RegEx perl    : [abcd]{3}\.[defg]{3}
 RegEx php     : [abcd]{3}\.[defg]{3}
 RegEx postgres: [abcd]{3}\\.[defg]{3}
 RegEx python  : [abcd]{3}\.[defg]{3}
 RegEx sed     : [abcd]\{3\}\.[defg]\{3\}
 RegEx vim     : [abcd]\{3}\.[defg]\{3}

     .oO(2452145)(¤abcd¤3¤.¤defg¤3)
  [1-9]:

followed by:
1) any character
2) a specific character
3) a literal string
4) an allowed characters list
5) a forbidden characters list
6) a special combination
7) a POSIX combination (locale aware)
8) a ready RegEx (not implemented)
9) anything
```

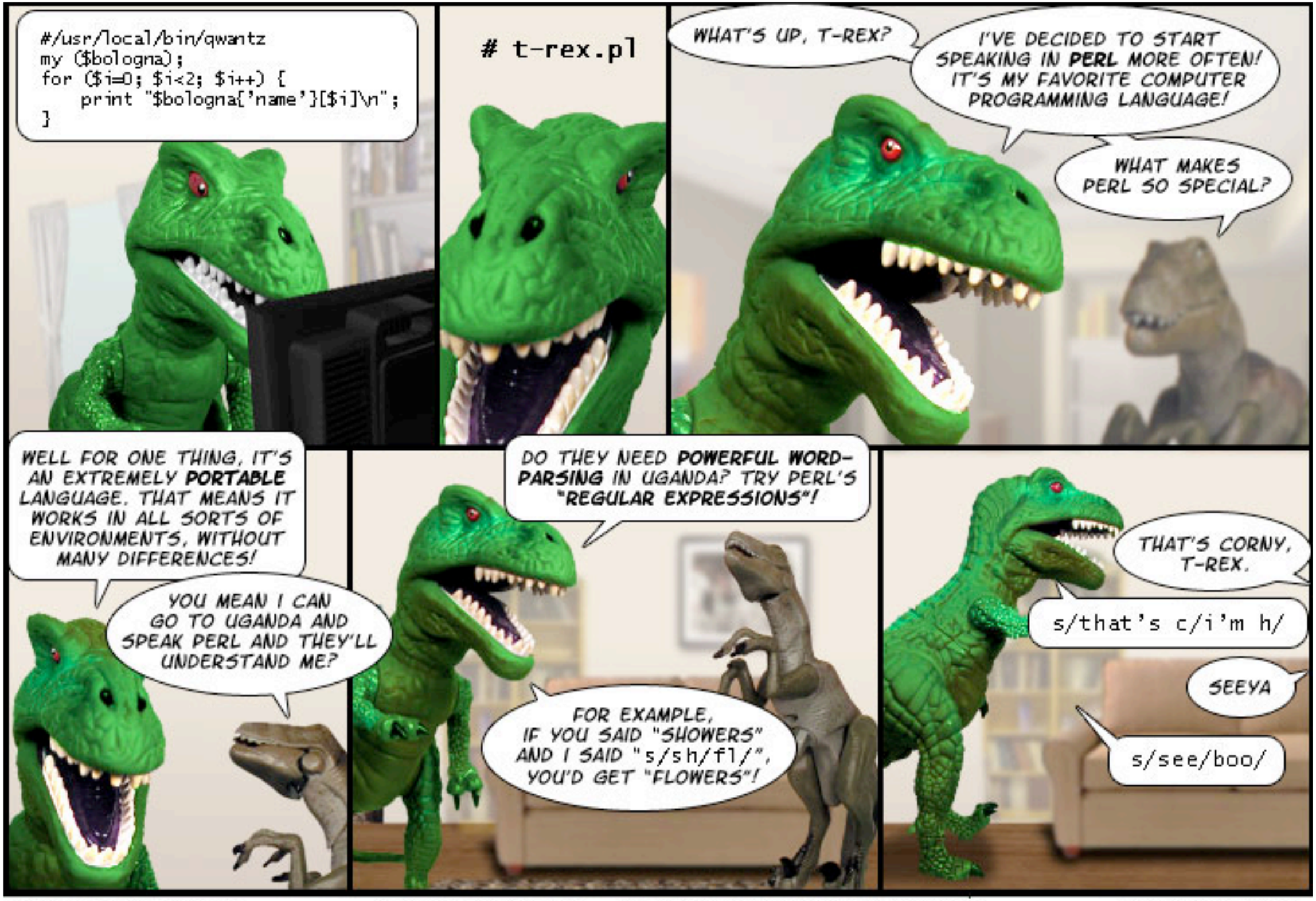
http://txt2regex.sourceforge.net

# “Uhm, Isn’t Perl Dead?”

- The CPAN continues to grow
- ACT Conferences
  - http://act.mongueurs.net/conferences.html
- Catalyst (MVC Web Framework)
  - http://catalyst.perl.org
- POE (Event Driven Programming Framework)
  - http://poe.perl.org
- DBIx::Class / Rose::DB (ORM)
- Duke Nukem Forever^W^W^W Perl 6
  - Rakudo*
- Moose (Real OO for Perl5)

See Schwern’s *Perl is Undead*

URL: **http://tinyurl.com/52ozwh**




http://www.qwantz.com/index.php?comic=658