regex

Matt Might

University of Utah

matt.might.net

source: xkcd

Today

- What's a regular expression?
- grep, egrep, sed, pinch of perl

Thursday

- awk rundown
- emacs, vim
- “SQL in the shell”

regex is a pattern language.

(a family of them, actually)

regex languages

- math
- unix
- pcre

Regexen match strings.

Design principles

Principle:

The frequent should be easy.

Principle:

Terse syntax aids coding.

Anti-pattern:

Make exception into rule.

Anti-pattern:

Take terse to the limit.

Anti-pattern:

Use both standards.

Anti-pattern:

Break the theory.

Anti-pattern:

Oops, a Turing machine.

Minimal regex

Pattern forms

Character: *c*

ex: a **matches** a

Sequence: $p_1 p_2$

ex: `ab` **matches** `ab`

Choice: $p_1 \mid p_2$

ex: `a|b` **matches** `a`

ex: `a|b` **matches** `b`

Repetition: $p*$

ex: a* matches a

ex: `a*` matches `aa`

ex: a* matches aaa

ex: a* matches

Subpattern: (p)

ex: `(ab)*` **matches** `ab`

ex: `(ab)*` **matches** `abab`

ex: `ab*` **matches** `abbbbb`

Sugared regex

Option: p?

ex: a? matches a

ex: a? matches

$$p? = (p|)$$

Strict repetition: $p+$

ex: a+ matches a

ex: a+ !matches

$$p+ = pp*$$

Char set: $[c_1 c_2 \ldots]$

ex: `[ab]` **matches** `a`

ex: `[ab]` **matches** `b`

Inverse char set: [^$c_1 c_2 \ldots$]

ex: `[^ab]` **matches** `c`

Warning: `echo $LANG`

Solution: `LANG=C`

Any character: .

ex: . matches a

ex: . matches c

ex: . matches b

Start of line/string: ^

End of line/string: $

Repeat: $p\{n\}$

ex: $p\{3\} = ppp$

Bounded repeat: $p\{n,m\}$

ex: $p\{2,4\} = pp|ppp|pppp$

Irregular expressions

Submatch: (p) and \1

ex: `(.)\1` **matches** `aa`

Shortest match: p+?

Dialects

Dialect: BRE

- `( )` => `\( \)`
- `{ }` => `\{ \}`
- `?` => `\?`
- `+` => `\+`
- `|` => `\|`

Dialect: BRE, ERE, PCRE

Defined char classes

BRE, ERE	PCRE
[[:word:]]	\w
[[:alpha:]]	[A-Za-z]
[[:space:]]	\s
[[:lower:]]	[a-z]
[[:upper:]]	[A-Z]
[[:digit:]]	\d
[[:punct:]]	

Define char classes

BRE, ERE	PCRE
\b	\b
[^[:space:]]	\S
[^[:word:]]	\W
[^[:digit:]]	\D

grep: examples and pitfalls

```
grep foo|bar words
```

```
grep 'foo|bar' words
```

```
grep 'foo\|bar' words
```

```
egrep 'foo|bar' words
```

```
grep '.x.n' words
```

```
grep '^.x.n$' words
```

```
grep '^(.*)\1$' words
```

```
egrep '^(.*)\1$' words
```

```
egrep '98.17.132.45' log
```

```
egrep '\b98\.17\.132\.45\b' log
```

```
egrep '([0-255]\.){3}[0-255]' log
```

```
egrep '\d|1?\d\d|2[0-4]\d|25[0-5]' log
```

```
grep '\d*[02468]' log
```

```
egrep '(0*(1(01*0)*1)*)*0*' log
```

```
perl -e 'while (<STDIN>) {
   if (/^1?$|^(11+?)\1+$/)
      { print }
}'
```

Challenge

- Match dates YYYY MM DD
- Account for leap years
  - If multiple of 400: Yes
  - If multiple of 100 but not 400: No
  - If multiple of 4 but not 100: Yes

Challenge

- Match RFC 3696 email addresses
- Never try this with lives at stake

Useful flags

- `-v` inverts the match.
- `--color` colors the matched text.
- `-F` matches literal string.
- `-H,-h` print (or not) filename.
- `-i` matches case insensitively.
- `-l` prints only filenames instead.

sed

stream editor

```
sed 's/regex/replacement/g'
```

But, sed is Turing-complete!

sed programs

```
#!/usr/bin/sed -n -f
command
address command
address1,address2 command
```

sed addresses

Line number: n

Last line: $

Regex address: `/regex/`

sed commands

s, q, p, d, #, {}, r, w, y

Warning! :, b, t, h, g, ...

s/*pat*/*rep*/*args*

s replacement

- & - fully matched text
- $\backslash n$ - text of nth submatch

s arguments

- g - global replace
- *n* - replace *n*th match
- w - write to file if match
- p - print if match

q

p

d

#

r *filename*

`w` *filename*

y/from/to/

sed examples

```
sed ‘10q’
```

```
sed 's/^/  /'
```

```
sed '/^#/d'
```

```
sed 'y/
abcdefghijklmnopqrstuvwxyz/
defghijklmnopqrstuvwxyzabc/'
```

```
sed -E \
's/([A-Z][a-z]*), ([A-Z][a-z]*( [A-Z][a-z]*[.]?)?)/\2 \1/g'
```

```
#!/usr/bin/sed -E -n -f
/<body>/,/<\/body>/ p
```

```
#!/usr/bin/sed -E -n -f
/<body>/,/<\/body>/ {
   /<body>/b
   /<\/body>/b
   p
}
```

```
#!/usr/bin/sed -E -n -f
/<body>/,/<\/body>/ {
   s/^.*<body>//
   s/<\/body>.*$//
   p
}
```

```
#!/usr/bin/sed -nf
s/./a/g
H
x
s/\n/a/
t a
: a;  s/aaaaaaaaaa/b/g; t b; b done
: b;  s/bbbbbbbbbb/c/g; t c; b done
: c;  s/cccccccccc/d/g; t d; b done
: d;  s/dddddddddd/e/g; t e; b done
: e;  s/eeeeeeeeee/f/g; t f; b done
: f;  s/ffffffffff/g/g; t g; b done
: g;  s/gggggggggg/h/g; t h; b done
: h;  s/hhhhhhhhhh//g
: done
$! {
  h
  b
}
: loop
/a/! s/[b-h]*/&0/
s/aaaaaaaaa/9/
s/aaaaaaaa/8/
s/aaaaaaa/7/
s/aaaaaa/6/
s/aaaaa/5/
s/aaaa/4/
s/aaa/3/
s/aa/2/
s/a/1/
: next
y/bcdefgh/abcdefg/
/[a-h]/ b loop
p
```

sed tetris

Design principle:

Allow abbreviations.

ex: `s` or `subst`

ex: `b` or `branch`

ex: `q` or `quit`

ex: `d` or `delete`

Questions?