Elements of Programming in Perl

<H16-12/13>

Regular Expressions: Pattern Matching and Substitution

Josep F. Abril

jabril@imim.es

Pattern Matching Operators and Funtions

Matching:

```
m/PATTERN/modifiers
```

Substitution:

```
s/PATTERN/REPLACEMENT/modifiers
```

Transliteration:

```
tr/SEARCHLIST/REPLACEMENTLIST/modifiers
```

Split:

```
@array = split /PATTERN/, $string
```

Grep:

```
@filtered = grep /PATTERN/, @array
```


The Binding Operator

Match operator not needed when using //

```
$var =~ m/PATTERN/   <==>   $var =~ /PATTERN/
```

Alternative delimiters

```
$var =~ m%PATTERN%
$var =~ m{PATTERN}
$var =~ s#PATTERN#REPLACEMENT#
$var =~ s{PATTERN}{REPLACEMENT}
```

Default variable to match

```
/PATTERN/   <==>   $_ =~ /PATTERN/
s/PATTERN/REPLACEMENT/   <==>   $_ =~ s/PATTERN/REPLACEMENT/
```

Evaluating as Conditions

```
$n = $var =~ m/PATTERN/;
if ($var =~ m/PATTERN/) { ... }; #    MATCH
if ($var !~ m/PATTERN/) { ... }; # MISMATCH

$n = $var =~ s/PATTERN/REPLACEMENT/;
if ($var =~ s/PATTERN/REPLACEMENT/) { ... };

$n = scalar ( split /PATTERN/, $string );

$n = grep /PATTERN/, @rray;
```


Regular Expression Metacharacters

Exact character:	`/atg/`
Any character:	`/./`
Quoting special chars:	`/\./`
Character class:	`/[acgt]/`
Negated char class:	`/[^0-9]/`
Anchors:	`/^...$/`
Alternate patterns:	`/t(ga|aa|ag)/`

More on Anchors

- `\A` Match only at beginning of string
- `\Z` Match only at end of string, or before newline at the end
  (`^` and `$` will match at every internal line boundary when the `/m` modifier is used)
- `\z` Match only at end of string
- `\G` Match only at pos() (e.g. at the end-of-match position of prior m//g)
- `\b` Match a word boundary
- `\B` Match a non-(word boundary)

Regular Expression Quantifiers

Asterisk -> 0 or more instances: `/atg*/`
`/atg*?/`

Plus -> 1 or more instances: `/atg+/`
`/atg+?/`

Question mark -> 0 or 1 instance: `/atg?/`
`/atg??/`

Braces specifying min and max instances:

`/atg{2,10}/`	2 through 10 "g"	`/atg{2,10}?/`
`/atg{2,}/`	2 or more "g"	`/atg{2,}?/`
`/atg{2}/`	exactly 2 "g"	`/atg{2}?/`

? ➔ Non-Greedy match operator

Greediness by Default

```
$s1 = $s2 = "I am very very cool";
$s1 =~ s/ve.*y //; # replace pattern with nothing
    # $s1 contains: I am cool
```

I␣am␣very␣very␣cool

Greedy match

/ve.*y␣/

```
$s2 =~ s/ve.*?y //; # replace pattern with nothing
    # $s2 contains: I am very cool
```

I␣am␣very␣very␣cool

Non-Greedy match

/ve.*?y␣/

Class Sequences

Whitespace / Non-whitespace

`\s` `[ \n\f\r\t]`
`\S` `[^ \n\f\r\t]`

Digit / Non-digit

`\d` `[0-9]`
`\D` `[^0-9]`

Word / Non-word

`\w` `[a-zA-Z0-9_]`
`\W` `[^a-zA-Z0-9_]`

Capturing with Special Vars

```
$dna_seq = "attgcatggaccgtaccgta";
$dna_seq =~ /gac*/;
```

Text BEFORE the match: $`
Matching pattern text: $&
Text AFTER the match: $'

```
print "Before match >>$`<<\n"; # "attgcatg"
print "       Match >>$&<<\n"; # "gacc"
print " After match >>$'<<\n"; # "gtaccgta"
```


Capturing with Parentheses

```
$dna_seq = "attgcatggaccgtaccgta";
$dna_seq =~ /((.gc).*(cg.))/;
```

$1 eq "tgcatggaccgt"
$2 eq "tgc" $3 eq "cgt"

```
$dna_seq =~ /(?:(.gc).*(cg.))/;
```

$1 eq "tgc" $2 eq "cgt"

```
$dna_seq =~ s/(ac)(.*)(t\1)/$2/;
```

$dna_seq eq "attgcatggcgcgta";

Extended Patterns

Comments:	(?#text)
Embeeded modifiers:	(?*imsx-imsx*:pattern)
Clustering:	(?:pattern) (?*imsx-imsx*:pattern)
Positive look-ahead:	(?=pattern)
Positive look-behind:	(?<=pattern)
Negative look-ahead:	(?!pattern)
Negative look-behind:	(?<!pattern)

Regexes and Sexeger

Regular expressions → REGEXES

```
# Match any occurrence of 19xx that's not followed by another one
if ($string =~ /(19\d\d)(?!.*19\d\d)/) {$date = $1}
# or, better
if ($string =~ /.*(19\d\d)/) {$date = $1}
```

Reversed regular expressions → SEXEGERs

```
$rev = reverse $string;
    # Reverse whatever string we were passed
$rev =~ /(\d\d91)/;
    # Look for the first occurrence of a xx91 (19xx reversed)
$found = reverse $1;
    # Reverse back whatever we found and return it
```

More info:

http://www.perl.com/pub/a/2001/05/01/expressions.html
http://japhy.perlmonk.org/sexeger/sexeger.html

Building Regular Expressions

Defining pattern within the operator

```
$num =~ m/^[-+]?\d+\.?\d*$/o;
```

Build patterns using scalar variables

```
$sign = '[-+]?';  $digits = '\d+';
$decimal = '\.?'; $more_digits = '\d*';
$number = "$sign$digits$decimal$more_digits";
$num =~ m/^$number$/; # don't use "o" modifier here
$num =~ m/^${number}$/;
    # better, specially when reusing it in other patterns
$num =~ m/^${number}Euros?$/;
```

Quoting pattern strings: quotemeta and \Q...\E

```
$string = '5.0^-12';
$quoted = quotemeta($string);
$num =~ m/$quoted/;   <==>   $num =~ m/\Q$string\E/;
```


Maintaining Regular Expressions

Using appropriate delimiters

```
s/\/usr\/local/\/usr\/share/g; # bad delimiter choice
s#/usr/local#/usr/share#g;    # better
```

Comments outside the regex

```
# turn the line into the first word, a colon, and
# the number of characters on the rest of the line
s/^(\w+)(.*)/ lc($1) . ":" . length($2) /meg;
```

Comments inside the regex (using the "x" modifier)

```
s{<                 # opening angle bracket
    (?:             # non-backreffing grouping paren
      [^>'"] *      # 0 or more things that are neither > nor ' nor "
        |           #    or else
      ".*?"         # a section between double quotes (stingy match)
        |           #    or else
      '.*?'         # a section between single quotes (stingy match)
    ) +             #   all occurring one or more times
  >                 # closing angle bracket
}{}gsx;             # replace with nothing, i.e. delete
```