A useful tool: regular expressions

Usage:
Validate, search or replace text (in DW: filter&clean data).

Appear in:

- PHP, javascript (validate input, reformat), SQL (pattern matching)
- script languages/UNIX scriptsp: `grep, sed, awk...`
- perl
- programming language libraries: `perl, python, java, c++...`
- parsers, packet analysis...
- text editors/IDE (`Find&Replace`))

Caveat:
Several "standards", features vary slightly. Main flavours:

- POSIX flavours: Basic and Extended (BRE, ERE)
- PCRE (originally from Perl).

Regex Engines:
PCRE, Oniguruma, RE2 (Google), Boost (C++), RegExp (Javascript)...

Regular expressions: memento

a	symbol a
`(r)`	`r` (delimiter/capture)
$r_1 \mid r_2$	r_1 or r_2 (alternative)
$r_1\, r_2$	concatenation

Special characters:

`.`	any symbol
`^`	text beginning
`$`	end of text

Quantification:

`r?`	0 or 1 occurrence of `r`
`r*`	0 or more occurrences
`r+`	1 or more occurrences
`r{`n`}`	exactly `n` occ.
`r{`n`,}`	at least `n` occ.
`r{`min, max`}`	between `min` and `max` occ.

Captured subexpression:

`\n` the substring matching n^{th} captured group

(defined by n^{th} opening parenthesis)

Character classes:

`[`$a_1 \ldots a_n$`]`	1 character: a_1 or a_2 or...
`[`a`-`d`]`	a, b, c or d
`[^...]`	any character *but* ...

Predefined character classes:

Posix	Description	PCRE
`[:alpha:]`	[A-Za-z]	
`[:alnum:]`	[A-Za-z0-9]	
`[:digit:]`	[0-9]	`\d`
`[:space:]`	[\t\r\n\v\f]=spaces	`\s`
`[:blank:]`	horizontal spaces	`\h`
`[:punct:]`	punctuation	
`[:upper:]`	uppercase	`\u`
`[:word:]`	[A-Za-z0-9_]	`\w`
`[:print:]`	[\x20-\x7E]=visibl char+ space	
`[=`a`=]`	equivalence class of "a"	

PCRE: uppercase to invert: `\D` = non-number.

Metacharacters for PCRE/ERE:

`^.[]$()*+?|{}\` échappés par \

Metacharacters for BRE:

`^.[]$*\`

Regular expressions: behavior

Algorithme de recherche des occurrences:

- by default, engine searches *first and longest* occurrrence
- anchors and assertions (^, $, \b, \B, (?=r)...) do not "match" symbols.

Character classes:

- POSIX charact classes used within "[]": ex: [[:alpha:]]
- equivalence classes [=a=] $\simeq$ [aàáâäåãAÀ...] determined by LC_CTYPE category in UNIX *locale*
- beware of digraphs, *é* may be 1 or 2 character, etc.
- including special charac in character classes:
  - *meta* status generally lost inside char class.
  - when - first or last in "[]" : no interval but symbol - itself.
  - classes cannot be empty nor nested, so brackets are matched in [a[b] and [ ]ab]

Captures:

- \0 captures whole string
- sometimes $n (outside pattern) instead \n (within) for backreference

Métacaractères:

- script/prog languages interpret pattern before forwarding to engine $\Rightarrow$ escape symbols ex: ( ) \ $\Rightarrow$ double escape!
- use preferably PCRE, or ERE: with BRE one must escape ( ){ }. And BRE has no alternation |, while \? \+ may be supported but are non-standard.

Regular expressions: examples

Examples (PCRE ou ERE):

- `[a-z]+0` matches text containing one or more lowercase followed by `0`.
- `^[0-9]{10}$` matches text (line) that is a 10-digit number.
- `[^0-9]{1,4}` matches: 12*ac*34589*:#@$*45
- `[[:alpha:]]*[[:digit:]][^[:alnum:],]+` matches: A#58*9:#@*,aa$b45
- `(ab|cd)+ee|^ab` matches: *ab*aacdz*cdabee*z
- `([a-c])z\1\1` matches *azaa* but not *azcc*.

  取前面已经出现的第一个
- Find:

  PCRE: `(\d{4})-(\d{2})-(\d{2})`
  ERE (POSIX classes): `([[:digit:]]{4})-([[:digit:]]{2})-([[:digit:]]{2})`
  ERE: `([0-9]{4})-([0-9]{2})-([0-9]{2})`
  BRE: `\([0-9]\{4\}\)-\([0-9]\{2\}\)-\([0-9]\{2\}\)`

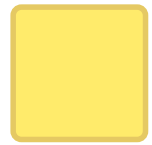
  Replace: `$3/$2/$1`

PCRE regular expressions: memento (advanced)

Misc:

分界

`\b`	word boundary (assertion, like `^`, `$`) and: `\<` start of word `\>` end)
`\B`	not a word boundary (anchor too)
`(?:r)`	non-capturing group



Assertions : ! if negative, = if positive, < for lookbehind

`(?=r)`	positive lookahead
`(?!r)`	negative lookahead
`(?<=r)`	positive lookbehind
`(?<!r)`	negative lookbehind

Conditional pattern (only in some engines: python, perl, pcre)

`(?(if)then|else)`

Named capture (pcre, python)

Capture	Reference	Replacement	
`(?P<GroupName>r)`	`(?P=GroupName)`	`\g<GroupName>`	Python, Perl, ...
`(?<GroupName>r)`	`\k<GroupName>`	`${GroupName}`	.Net, Java
`(?<GroupName>r)`	`\k<GroupName>`	`$+{GroupName}`	Perl

Backreferences and replacements can also use the group's number, ex: `\1`.

Options (non standard, but widely supported, one way or another)

`i`	case-*i*nsensitive
`m`	*m*ultiline: if text has symbols, `^` `$` match line extremities
`s`	*s*ingle-line: enable "." to match newline char
`x`	e*x*panded: spaces ignored unless escaped...

PCRE Regular expressions: examples

Examples:

- `new(?!s)` sur "Those news seem *new*er than *new*"
- `(?ms)^a(.)*z$` sur "*abcd*\n*gfz*\na"
- regexp for passwords ($\geq$ 8 symbols, digits, punctuation, uppercase) ?

- `([a-c])z\1\1` équivaut à `(?P<lettre>[a-c])z\1\1` et à `(?<le>[a-c])z\k<le>\k<le>`
- Find: `(?P<annee>\d{4})-(?P<mois>\d{2})-(?P<jour>\d{2})`
  Replace: `\g<jour>/\g<mois>/\g<annee>`

regular expressions: greedy, lazy, possessive quantifiers:

Greedy, lazy/reluctant, possessive quantifiers:

By default quantifiers are *greedy*: from a position, match as many occurrences as possible, then backtrack if no solution for global pattern.

- With `?` quantifier becomes *lazy*: the fewest occurrences, then increases if no solution.
- With `+` quantifier becomes *possessive* (`Java, Python, Perl...`): max occurrences, no backtracking even if it fails.

Examples:

- `ba*` over "a*baaa*c"
- `ba*ac` over "a*baaac*"
- `ba+?` over "a*ba*aac"
- `ba+?c` over "a*baaac*"
- `ab{2,}+[a-z]` over "a*abbc*"
- `ab{2,}+[a-z]` over "aabbb"
- `([a-c])*+cz` can never match.

Regular expressions under UNIX: `grep`, `sed`. BRE by default, but ERE with option -E

- `egrep` = `grep -E` : in input file(s), returns lines having a match.
  - -i case insensitive
  - -n displays line numbers
  - -R (recursive) all files in input directory
  - -l displays filenames only (hence stops at 1ère match per file)
  - -a searches binary files as if they were text
  - -A / -B / -C displays lines around the match
  - --include / --exclude / --exclude-dir specifies searched files: useful combined with -R.

- `sed` is a utility that modifies text.
  - -E ERE instead of BRE (for Mac&GNU), -r (GNU)
  - -i modifies input file (in-place)
  - -e must be written before each action if there are several (or `-f scriptfile` : script provided in a file)
  - . . . many other options

```
sed -e 's/before/after/g' infile.txt > outfile.txt
```

. . . replaces every occ of before by after.

Exemples:

- `egrep 'ion$' /usr/dict/words` . . . returns words ending in *ion*.
- `grep -rE --color='auto' '\best\b' Desktop/` . . . searches word *est* in Desktop.
- `grep -Ein -B4 --color='auto' 'port' Desktop/fichier.txt`
  . . . searches port (possibly uppercase), displays 4 previous lines, numbered
- `find / -type f -exec grep -l 'motif' {} \;` (combined with `find` to specify the files)
- `sed -Ei.back 's#([0-9]{4})-([0-9]{2})-([0-9]{2})#\3/\2/\1#g' a.txt`
  . . . modifies in-place the date format, with backup.

More powerful utilities than `sed` and `grep`: PCRE syntax

- `ack`: more convenient to use than `grep` (voir `https://beyondgrep.com/`):
  - same regexes as `perl`
  - by default: search is recursive and restricted to code files (not binaries, .git...)
  - highlightning
- `ag`: similar to `ack` (clone), claims to be faster
- `awk`: probably more convenient than `sed` if file is structured.
- `perl`:

Examples:

- `perl -pi -e 's/a/b/g' file.txt` similar to `sed -i -e ...` †
- `perl -i.back -pe 's/(?<annee>\d{4})-\d{2}-\d{2}/$+{annee}/g' file.txt`

  ...but in `perl` or `awk` we can use function `strftime`
- `perl -ne '/\B\u/ && print' file.txt` similar to `grep -e ...` †

- `awk -F',' '{print $2}' file.txt` returns 2nd column of a csv
- `awk -F';' '/^[0-5;]*$/{print $9}' f.txt` the 9th col. of lines satisfying the regex.

† but PCRE syntax

Java regular expressions

- Java:

```
import java.util.regex.*;  //Matcher, Pattern
public class Regexptest
{
  public static void main(String[] args)
  {
  System.out.println(Pattern.matches("\\w* mops", "with 7 mops")); //false

  Pattern p = Pattern.compile("([a-z ]*)\\s*side");
  Matcher m = p.matcher("from side to");

  while (m.find( )) {
    System.out.println("Line: " + m.group(0)); //"Line: from side"
    System.out.println("Value: " + m.group(1)); //"Value: from "
    System.out.println(m.start() + " " + m.end()); //0 9
  }

  System.out.println(m.matches()); //false: should match entire region

  System.out.println(m.replaceAll("z")); //z to
  }
}
```

No support for classes: `\l` `\L` `\u` `\U`. No support for conditions either.

Scala and Python regular expressions

- Scala: relies on Java library.

```scala
import scala.util.matching.Regex

val motif: Regex = """(\d\d\d\d)-(\d\d)-(\d\d)""".r // .r turns into regexp
val dates = "historique: 2004-01-20, 1958-09-05, 2010-10-06, 2011-07-15"

val allYears = for (m <- motif.findAllMatchIn(dates)) yield m.group(1)
// iterator: allYears.next() returns successively "2004" , "1958" , "2010"

val YearsOnly = motif.replaceAllIn(dates, m => m.group(1))
// history: 2004, 1958, 2010, 2011
```

- Python: functions `match`, `search`, `findall`, `sub`.

```python
import re
c = re.search('(\d\d?) \w+ \d{4}', 'le  16 avril 2017')
print c.group(1) # 16

motif = re.compile('\d\d? \w+ \d{4}')
c = motif.search('le  16 avril 2017')
print c.group()  # '16 avril 2017'

re.sub('([0-9]{4})-([0-9]{2})-([0-9]{2})','\\1/\\2/\\3','2016-04-16')
# '2016/04/16'
```

SQL regular expressions

- Oracle : POSIX ERE compliant.

```
UPDATE countries
  SET name =  REGEXP_REPLACE(name,'(.)','\1 ') WHERE name != France;
-- name becomes: B r a z i l
SELECT first_name, last_name FROM employees
  WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$')
```

- PostgreSQL : implements regexp-like patterns with SIMILAR TO, and supports some POSIX regexp functions:

```
SELECT col  FROM t WHERE (col similar to '%(b|d)%');
-- returns "abc", but not "aca"
SELECT regexp_replace('foobarbaz', 'b..', 'X', 'g')
--  fooXX
```

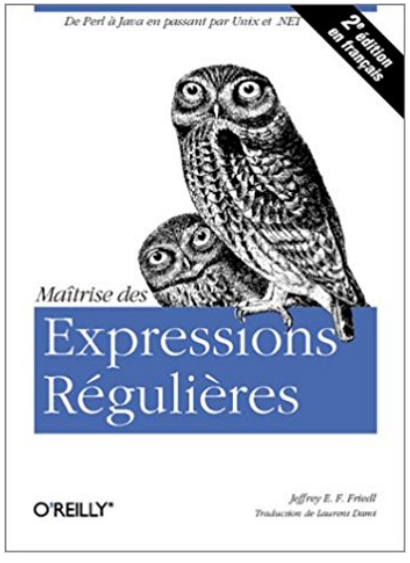
- MySQL : Herbert Spencer's regex library (POSIX)
- MariaDB : PCRE library (prev versions: regex)
- Microsoft SQL Server : partial support with LIKE (afaik)
- DB2 : no direct support (afaik) ⇒ UDF.

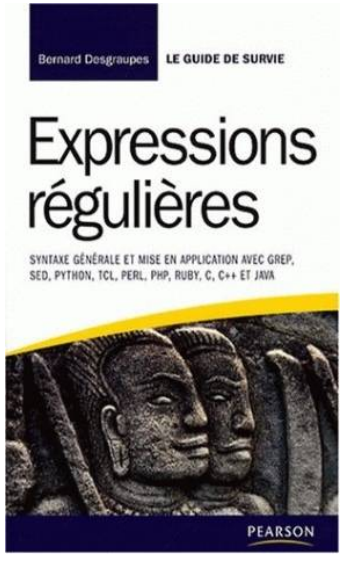
Can also call regexp library through UDF.

References

- http://www.rexegg.com/regex-quickstart.html (comprehensive)
- https://en.wikipedia.org/wiki/Regular_expression
- http://www.regular-expressions.info
- http://www.expreg.com/presentation.php (comprehensive)
- https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/les-expressions-regulieres-partie-1-2
- https://stackoverflow.com/questions/22937618/reference-what-does-this-regex-mean
- https://regex101.com (test/debug a regex)
- http://www.catonmat.net/blog/perl-one-liners-explained-part-six/ (perl one-liners)

À la B.U. (Paris-Sud) :





`http://www.regular-expressions.info/books.html` includes some book reviews.