Regular expressions

Regular expressions

- Regular expression, regex or regexp is a formal language theory, a sequence of characters that define a search pattern.
  This pattern is usually used by string searching algorithms for „find“ or „find and replace“ operations on strings.
  (source: Wikipedia)

- Different tools which use regex: Google Code Search (shut down in March 2013), grep, IDE code competition, Kate, ed, find, locate, vi, emacs, .NET, Java SDK, Exalead, etc.

Basic concepts

- Every sign represents itself **except** [\^$.|?*+(){}
- \ followed by *[\^$.|?*+(){}* represents the followig sign
- *[str]* represents **one** sign from the set in the brackets: *s*, *t* or *r*
- *[^str]* represents negation of the set – sign which is **not** *s*, *t* or *r*
- - (except before or after [) represents a range: *[a-zA-Z0-9]* all numbers and characters
- *[-]* represents a *minus*
- *(str)* represents a *substring/subexpression* which can be recalled later
- | represents *a choice* (also known as alternation or set union): *a|b* means *a or b*; *a(b|c)d* means *abd or acd*

Basic concepts

- . matches *any single character* (except space or new line):
  a.cd can be *abcd, aXcd,...*
- *[.]* represents a *fullstop*
- * after a sign means **zero or more** repretitions of the sign:
  *ab*c* can be: *ac, abc, abbc,...*
  *a(bb)*c* can be: *ac, abbc, abbbbc,...*
  *[xyz]** can be *‘‘, x, y, zx, zyx,...*
- *+* after a sign means **one or more** repetitions of the sign:
  ab+c can be: *abc, abbc,...*
- *[xyz]+* can be: *x, y, zx, zyx,...*
- *?* means previous sign (string) can be present or not: *ab?c* can be *ac or abc*

Basic concepts

- *{}* limiting the number of repetitions of a previous sign (subset)
  {n} repetition of a previous sign *exactly n-times*: *a{3}* means *aaa*
  {n,m} repetition of a previous sign *at least n-times and at most m-times*
  {n,} repetition of a previous sign *at least n-times*
- ^ represents **start** of a string
- *$* represents **end** of the string

Abbreviations

POSIX	ASCII	Meaning
[:upper:]	[A-Z]	Uppercase letters
[:lower:]	[a-z]	Lowercase letters
[:alpha:]	[A-Za-z]	Alphabetic characters
[:alnum:]	[A-Za-z0-9]	Alphanumeric characters
[:digit:]	[0-9]	Digits
[:xdigit:]	[A-Fa-f0-9]	Hexadecimal characters
[:punct:]	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
[:blank:]	[[\t]]]	Space and tab
[:space:]	[\t\r\n\v\f]	Whitespace characters
[:cntrl:]	[\x00-\x1F\x7F]	Control characters
[:graph:]	[\x21-\x7E]	Visible characters
[:print:]	[\x20-\x7E]	Visible characters and the space character

Example

- *(a|b)*ccc*
  represents strings which can start with ***any number of*** repetitions of a letter ***a*** and ***any number of*** repetitions of a letter ***b*** and end with ***three repetitions of*** a letter ***c***.
  - Sign **|** separates alternatives: *a*|*b* means „*a* or *b*“
  - Sign ***** means ***zero or more*** repetitions of the expression before the sign: *(a|b)** means „any number of repetitions of *a* or *b*“
  - (*subset*): brackets are used to limit the subset
  - At the end there are 3 repetitions of a letter *c*

ccc	cccccc (wrong!)	bbbbaccc
ababaaccc	cccaababa (wrong!)	acccc

Example

- (Luis Fonsi)|(luis fonsi)

  Luis Fonsi
  luis fonsi

- (L|l)uis (F|f)onsi

  Luis Fonsi
  Luis fonsi
  luis Fonsi
  luis fonsi

- (a*)b(a*)b(a*)b(a*)

  All strings of *a*-s and *b*-s where *b* is repeted exactly three times

Example

- 0|((1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*)

  String 0 and all strings of digits which don‘t start with 0.
- 0|([1-9][0-9]*)

  String 0 and all strings of digits which don‘t start with 0.
- [A-Z][a-z]*

  All strings of letters which start with a capital letter.
- [A-Da-z]*

  Strings which contain letters A, B, C and D and small letters, example: aaaBfdCDsdfsdAzz.

Exercise

- On the desktop go to the directory we have created last time
- **Create** new **directory** bbb.txt

- **Move to** the directory bbb.txt

- Use editor **emacs** and create a file names.txt
  - How? Try typing in emacs names.txt

- Check what is stored in a file names.txt

Exercise

- Open file names.txt with **nano** editor and add name Vida to the file, save and close it.

- Check what is written to the file names.txt

- Use regular expressions and **egrep** command to find all starting lines in a file names.txt.
  - How does **egrep** command work? How do we figure that out?

Exercise

- Find all lines in the file names.txt starting with **M**.
- Find lines **starting with M and ending with a** with any number of letters in between.
- Find lines ending with a.
- Lines ending with a and have at least three letters before the a.

Exercise

- Lines ending with a and have exactly 4 letters.
- Lines which start with either M or L
- Write lola to the end of a file names.txt without using the editor.
- Write LOLA to the end of a file names.txt without an editor.
- Show the content of a file to the screen.

Exercise

- Print out all lines containing name lola where every letter can be a small or a big letter.
- Lines starting with Marjan and have 0 or more letters after it.
- Lines starting with Marjan and have at least one letter after it.
- Lines starting with Marjan and have one or 0 letters after it.

Exercise

- Print the detailed content of the directory bbb.txt.
- Print out detailed content of the directory bbb.txt where the results finish with .txt
- What does **locate** command do?
- Use locate command and regular expression to find all files finishing with .txt.