

# The ABC of computational Text Analysis

## 06: Learning Regular Expressions

Alex Flückiger  
2 April 2020

# Recap last Lecture

- counting words
  - particular or all words*
- preprocessing steps

# Outline

- discussion assignment
- introduction regular expression 
- Practicing RegEx

# Assignment #1



Sorry,  
this was too steep!

# Feedback Assignment #1

## example solution



- great dedication
- creative solutions I didn't show



- make sure folder exists  
*check with `ls` or in GUI*
- Home on Windows  
*`/mnt/c/Users/YOUR_WINDOWS_USERNAME/`*
- use operators `>` `>>` `|` anytime  
*files don't need to preexist*
- wildcard `*` saves you work  
*`*2015*.txt`*

# Review beyond the Shell



- too brief to be clear
- missing parts  
*feedback, second task*



- `locate` didn't work

# Writing runnable Scripts

## Requirements

- use text editor, not Word
  - Mac: TextWrangler*
  - Win: Notepad++*
- no prompt, just commands
- # precedes any non-commands
- file suffix `.sh`

The beauty of scripting is automation. 

# Assignment #2

- get/submit via OLAT
  - starting tonight*
  - deadline 16. April 2020, 23:59*
- use forum on OLAT
  - subscribe to get notifications*
- ask friends for support, not solutions

# Questions

# Text as Pattern

# Generalized Pattern

Problem



How to find **any** email addresses in a document?

```
Please contact us via info@organization.org.  
For specific questions ask Mrs. Green (a.green@mail.com).
```

Solution



Writing a generalized pattern

```
[\w._-]+@[\\w._-]+ # matches any valid email address
```

# What are patterns for?

- finding
- extracting
- removing
- replacing

... any textual parts

# What is RegEx?

RegEx = Regular Expressions ~ Patterns

- fixed string of characters  
*word, phrases, dates etc.*
- abstract, highly flexible expressions  
 $[Cc]o+1$  → *Col, col, Cool, coool ...*  
*literal + meta-characters*
- akin to regular languages

# Finding + Extracting

globally search for regular expression and print (grep)

- tool to filter to keep certain lines only
- allow extended regex patterns

*use `egrep` instead of `grep`*

```
egrep 'yes' file.txt          # search in a specific file
egrep -r 'yes' folder         # search recursively within folder
egrep 'yes'                  # keep lines containing pattern (yes)
egrep -i 'yes'                # do NOT ignore casing (Yes, yes, YES, ... )
egrep -v 'noisy'              # do NOT keep lines containing noisy
# prepare for counting by extracting raw match only
egrep -o 'only'               # print match only, not entire line
egrep -h 'only'                # suppress file name
```

# Quantifiers

repeat preceding character X times

- `?` zero or one
- `+` one or more
- `*` zero or any number
- `{n}`, `{m, n}` a specified number of times

```
egrep -r "Bundesrath?es"  
egrep -r "aa+"  
egrep -r "e{2}"
```

# matches old and new spelling of  
# matches two or more "a"  
# match sequences of three vowels

# Character Sets

- `[...]` any of these character
  - any vowel:* `[aueoi]`
  - any digit:* `[0-9]`
  - any letter:* `[A-Za-z]`
- `[^...]` any character but none of these
  - anything but the vowels:* `[^aueoi]`

```
# match the capitalized and non-capitalized form
egrep -r '[Gg]rüne'

# match sequences of 3 vowels
egrep -r '[aeiou]{3}'

# extract all bigrams (sequence of two words)
```

# Special Symbols

- `.` any character (excl. newline)
- `\` escaping to match literal
  - `\.` means the actual `.`, not “any symbol”*
- `\w` any alpha-numeric character
  - same as `[A-Za-z0-9_]`*
- `\s` any whitespace (space, newline, tab)
  - same as `[ \t\n]`*

# The power of ...

`.`\* matches *any character any number of times*

# More complex Examples

```
# extract websites
egrep -ro "www\.\w+\.\w+"
# extract emails
egrep -ro "[\w\.-]+\@[ \w\.-]+\*\*/\*.txt
```



Start simple,  
add complexity

# Replacing + Removing

stream editor (sed)

- advanced find + replace using regex

*sed "s/WHAT/WITH/g" FILE*

- `sed` replaces any sequence, `tr` only single symbols

```
echo "hello" | sed "s/llo/y/g"          # replace "llo" with a "y"  
# by setting the q flag in "s/llo/y/g", not only the first one
```

# Contextual Replacing

- reuse match with grouping
  - $\backslash 1$  equals the expression inside first pair of parentheses
  - $\backslash 2$  expression of second pair

...

```
# swap order of name (last | first -> "first" | last)
echo "Lastname Firstname" | sed -E "s/(.*)(.*)/\2 \1/"
# matching also supports grouping
# match any pair of digits (two identical digits)
egrep -r "([0-9])\1([0-9])\2"
```

# Something actually useful

combining regular expressions with frequency analysis

```
# count political areas, by looking up words ending with "politik"  
egrep -rioh '\w*politik' *.txt | sort | uniq -c | sort -h  
# count ideologies/concepts by looking up words ending with -ismus  
egrep -rioh '\w*ismus' *.txt | sort | uniq -c | sort -h
```

# In-class: Exercises I

1. Update your local copy of the Github repository KED2020 with `git pull`. Go to the party programmes in `materials/party_programmes.txt`.
2. Use `egrep` to extract all uppercased words that are abbreviations in most cases (e.g., UNO, SVP, SP).
3. Use `egrep` to extract words following any of these strings: `der die das`.
4. Do the self-check on the next slide.
5. Use `sed -E` to remove the table of content, the footer and the page number in the programme of the Green Party. Check the PDF to get a visual impression and test your regular expression with `egrep` to see if you match the correct parts in the document.

# In-class: Self-Check

equivalent patterns

$a^+ == aa^*$   
 $a? == (a| )$   
 $a\{3\} == aaaa$   
 $a\{2,3\} == (aa|aaa)$   
 $[ab] == (a|b)$   
 $[0-9] == (0|1|2|3|4|5|6|7|8|9)$

# "a" once or more than once  
# "a" once or nothing  
# three "a" or "a"  
# two or three "a"  
# "a" or "b" # any digit

# In-class: Exercise II

1. Since you know about RegEx, we can use a more sophisticated tokenizer to split a text into words. What is the difference between the old and new approach? Test it and check the helper page with [man](#).

```
# new, improved approach
cat text.txt | tr -sc "[a-zAÖÜöü-9-]" " \n"
# old approach
cat text.txt | tr ' ' '\n'
```

# More Resources

required

- Ben Schmidt. 2019. Regular Expressions. [online](#)

highly recommended

- Nikolaj Lindberg. egrep for Linguists. [online](#)