

The ABC of Computational Text Analysis

#5 BASIC NLP WITH THE COMMAND-LINE

Alex Flückiger
Faculty of Humanities and Social Sciences
University of Lucerne

27 March 2025

Recap last lecture

- understand the filesystem 

`/home/superhero/documents/text_analysis_2025` with `script.py` and `data/dataset.csv`

absolute path to dataset:

`/home/superhero/documents/text_analysis_2025/data/dataset.csv`

relative path to dataset from script:

`data/dataset.csv`

- perform shell commands

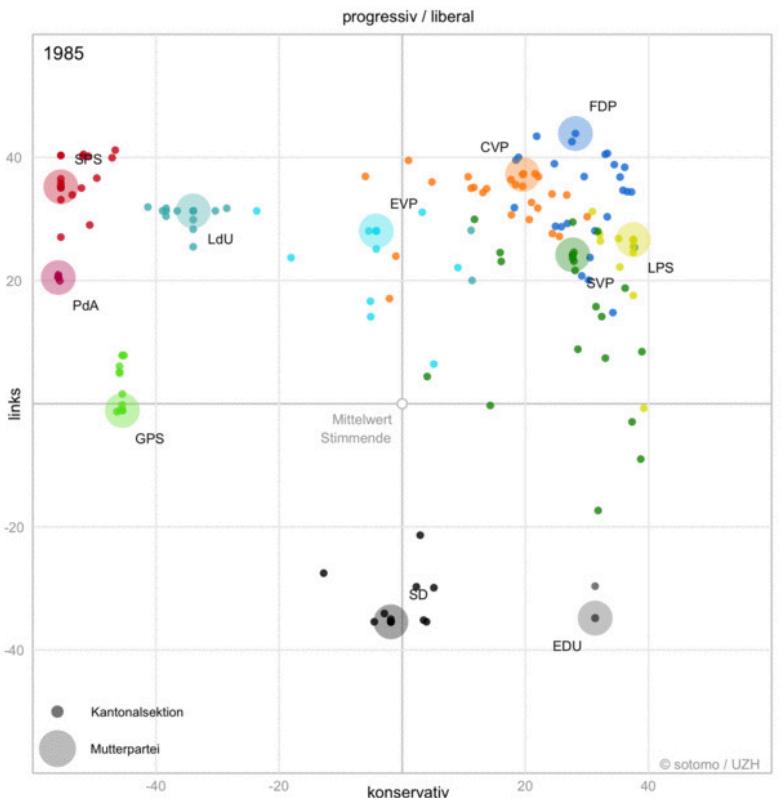
create/copy/move/remove files 

pipe output into other command 

Outline

- performing corpus linguistic using the shell 
counting, finding, comparing
- analyzing programmes of Swiss parties 

When politics is changing, language is changing.



Historical development of Swiss party politics ([Tagesanzeiger](#))

How to process a text collection

1. data in any format (e.g., `.pdf`)
2. each document as individual file (`.txt`)
use Shell for quick analysis
3. a dataset of documents (`.csv`, `.tsv`, `.xml`)
use Python for in-depth analysis



Counting Things

Measuring relevance by frequency

Bag of words approach

- counting words regardless of context
- simple (and simplistic)
- powerful
- fast



Representation of text as a bag of words

Get key figures of texts

```
wc *.txt      # count number of lines, words, characters
```

Get all word occurrences

Show phrase in context

```
egrep -ir "data" FOLDER/      # search in all files in given folder

# common egrep arguments:
# -i          search case-insensitive
# -r          search recursively in all subfolders
# --colour    highlight matches
# --context 2 show 2 lines above/below match
```

Count word occurrences

Get term counts per file

```
egrep -ci "big data" *.txt # search in file of type .txt only  
# arguments:  
# -c count matches per file  
# -i match case-insensitive
```

Word frequencies

Steps of the algorithm

1. split text into one word per line (tokenize)
2. sort words alphabetically
3. count how often each word appears

```
# piping steps to get word frequencies
cat text.txt | ... > wordfrq.txt
```

Frequencies of all words

Steps of the algorithm

1. split text into one word per line (tokenize)
2. sort words alphabetically
3. count how often each word appears

```
# piping steps to get word frequencies
cat text.txt | tr " " "\n" | sort | uniq -c | sort -h > wordfrq.txt

# explanation of individual steps
tr " " "\n"      # replace spaces with newline
sort -h          # sort lines alphanumerically
uniq -c          # count repeated lines
```

Convert stats into dataset

- convert to `.tsv` file
- useful for further processing
 - e.g., import in Excel

```
# convert word frequencies into tsv-file
# additional step: replace a sequence of spaces with a tabulator
cat text.txt | tr " " "\n" | sort | uniq -c | sort -h | \
tr -s " " "\t" > test.tsv
```

Two kinds of word frequencies

- absolute frequency
= `n_occurrences`
- relative frequency
= `n_occurrences / n_total_words`
allows to compare corpora of different sizes

In-class: Matching and counting I

1. Print the following sentence in your command line using echo.

```
echo "There are related computational fields: \
NLP, Computational Linguistics, and computational text analysis."
```

2. How many words are in this sentence? Use the pipe operator | to pass the output above to the command wc.
3. Match the word computational and colorize its occurrences in the sentence using egrep.



Continue on the next slide.

In-class: Matching and counting II

4. Get the frequencies of each word in this sentence using `tr` and other commands.
5. Save the frequencies into a tsv-file, open it in a spreadsheet programm (e.g., Excel, Numbers) and compute the relative frequency per word.
6. Are there some words that, although different, should be considered as the same?

Preprocess your data

refine the results with

- lowercasing
- replacing symbols
- joining lines
- trimming header + footer
- splitting into multiple files
- patterns to remove/extract parts

Text as Pattern

Searching with patterns

What about **abstract matches** instead of exact matches?

How to find all words starting with the letter **A**?

```
egrep -r "A\w+" **/*.txt
```

How to match any numerical dates?

```
egrep -r "[0-9]{1,2}\.[0-9]{1,2}\.[0-9]{4}"
```

When using patterns?

- finding 
- extracting 
- removing/cleaning 
- replacing 

... specific parts in texts

An introduction to Regular Expressions (RegEx)

RegEx builds on two classes of symbols

- **literal** characters and strings
 - letters, digits, words, phrases, dates etc.
- **meta** expressions with special meaning
 - e.g., `\w` represents all alphanumeric characters

Combining RegEx with frequency analysis

Something actually useful! 😊

```
# 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

# arguments:
# -o    ouput only match and not entire line
```

Quantifiers

Repeat preceding character X times

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

```
egrep -r "a+"          # match one or more "a"  
egrep -r "e{2}"         # match sequence of two "e"
```



Do not confuse regex with Bash wildcards!

Character sets

- [. . .] any of the characters between brackets
 - any vowel: [auoei]
 - any digit: [0-9]
 - any uppercased letter: [A-Z]
 - any lowercased letter: [a-z]

```
egrep -r "[Gg]rüne" # match both `grüne` and `Grüne`  
egrep -r "[aeiou]{3}" # match sequences of 3 vowels
```

Special symbols

- `.` matches any character (excl. newline)
- `\` escapes to match literal
 - `\.` means the literal `.` instead of “any symbol”
- `\w` matches any alpha-numeric character
 - same as `[A-Za-z0-9_]`
- `\s` matches any whitespace (space, newline, tab)
 - same as `[\t\n]`

```
# match anything between brackets
egrep -r "\(.*\)"
```

The power of . * 💪

Match *any character any times*

In-class: Matching with RegEx

1. Go to the website <https://www.swissinfo.ch/ger> and copy a few paragraphs of a random article.
2. After that, go to the website <https://regex101.com/> and paste the text into the big white field.
3. Write various regex patterns in the small field to match.
 - all uppercased words (~nouns)
 - words with exactly 5 characters
 - all words at the beginning of a sentence (~after a period)
 - match everything between quotes
4. Come up with your own challenge.



Questions?

Additional resources for using Bash

When you look for useful primers on Bash, consider the following resources:

- [Tutorial Basic Text Analysis by W. Turkel](#)
- [Tutorial Pattern Matching + KWIC by W. Turkel](#)
- [Cheatsheet of this course](#)

Additional resource for using Regex

- online regular expression editor [regex101](#)
to write and check patterns
- interactive regex tutorial [regexone](#)
- introductions to regular expressions
 - [A Visual Guide to Regular Expression](#) (highly recommended)
 - [Regular Expressions](#) (comprehensive; for digital humanities)

Optional exercise: Get the programmes of Swiss parties

1. Change into your local copy of the GitHub course repository KED2025 and update it with `git pull`. When you haven't cloned the repository yet, follow section 5 of the [installation guide](#).
2. You find some party programmes (Grüne, SP, SVP) in `ked/materials/data/swiss_party_programmes/txt`. Change into that directory using `cd`.
3. The programmes are provided in plain text which I have extracted from the publicly available PDFs. Have a look at the content of some of these text files using `more`.

Optional exercise: Analyzing Swiss party programmes

1. Compare the absolute frequencies of single terms or multi-word phrases of your choice (e.g., Ökologie, Sicherheit, Schweiz)...

across parties

historically within a party

Use the file names as filter to get various aggregation of the word counts.

2. Pick terms of your interest and look at their contextual use by extracting relevant passages. Does the usage differ across parties or time?