# Natural Language Processing

Various Text Processing Tools

## Linux Tools

- Linux contains various command line tools for text processing, e.g.:
  - grep
  - sed
  - awk
  - sort
  - uniq
  - head, tail

## What about Windows?

- In Windows you can install Cygwin http://www.cygwin.com/
  - Cygwin is a collection of tools which provide a Linux look and feel environment for Windows.

## grep

- A utility for searching plain-text data sets for lines matching a regular expression
- grep = **G**lobal **R**egular **E**xpression **P**rint
- Example:
- `grep 'ab*c' myFile`
  - Prints all the lines from the file myFile containing the strings ac, abc, abbc, abbbc, etc.
- grep tutorial： http://www.uccs.edu/~ahitchco/grep/

## sed

- A utility that parses and transforms text.
- sed = **S**tream **Ed**itor
- Great for “search and replace”
- Example:
- `sed 's/oldstuff/newstuff/g' input > output`
  - Substitutes the string (regex) `oldstuff` with `newstuff` (globally) in all lines in the file `input` and writes the result to file `output`
- sed tutorial: http://www.grymoire.com/Unix/Sed.html

# awk

- A scripting programming language typically used as a data extraction and reporting tool.
- awk= Alfred **A**ho, Peter **W**einberger, Brian **K**ernighan
- "**AWK** is a language for processing text files. A file is treated as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields, so we can think of the first word in a line as the first field, the second word as the second field, and so on. An AWK program is of a sequence of pattern-action statements. AWK reads the input a line at a time. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed." **Alfred V. Aho**

## awk

- awk tutorial: http://www.grymoire.com/Unix/Awk.html
- Example:
- `awk '{print $1"\t"$3}' input > output`
- Prints to file `output` the first field (column) followed by a tab character, followed by the third field from the file `input`

## sort and uniq

- Let us assume file `input` contains one token per line
- Counting frequencies:
- `sort input | uniq -c | sort -nr > output`
    - The result is a *unigram* model

## head and tail

- `head -3 < input`
  - Returns the first three lines
- `tail -2 < input`
  - Returns the last two lines
- `tail --lines=+2 < input`
  - Skips the first line

## Building a bigram model

- Let us assume that the file `eng.tok` contains one token per line.
- `tail --lines=+2 < eng.tok > eng2.tok`
- `paste eng.tok eng2.tok > eng.bigrams`
- `sort eng.bigrams | uniq -c | sort -nr > eng.freq`

## Lexical Analyser

- A lexical analyzer (í. lesgreinir) is a program which breaks a text into tokens (lexemes).
- A program which generates a lexical analyser is called a *lexical analyser generator* (í. lesgreinissmiður)
- Examples: Lex/Flex/JFlex
  - The user defines a set of regular expression patterns.
  - The program generates a finite-state automata.
  - The automata are used to recognise tokens.

## JFlex (http://jflex.de/)

- A tool which generates a lexical analyser given a set of regular expressions.
  - Generates Java code, which contains a finite-state automaton (state transition table).
- **Input**: JFlex source program (e.g. Simple.flex)
- **Output**: Java code (e.g. Simple.java)
- The Java code is compiled and exectuted
  - javac Simple.java (the output is Simple.class)
  - java Simple <textfile>

# JFlex

- To make JFlex run in Windows:
- Set c\:jflex\bin into path.
- Change the file c:\jflex\bin\jflex.bat to:

```
set JFLEX_HOME="C:\JFLEX"
REM for JDK 1.2
java -Xmx128m -jar %JFLEX_HOME%\lib\JFlex.jar
```

# JFlex example

```
%% A finite-state automata recognising (a|b)*abb
%public
%class Simple
%standalone
%unicode
%{
        String str = "Found: ";
%}

Pattern = (a|b)*abb
%%

{Pattern} { System.out.println(str + " " + yytext());}
. { ;}
```

# JFlex example

```
%% A good tokeniser for English?
%public
%class EngGood
%standalone
%unicode
%{
%}

WhiteSpace = [ \t\f\n]
Lower = [a-z]
Upper = [A-Z]
EngChar = {Upper}|{Lower}
EngWord = {EngChar}+
%%
{WhiteSpace} {;}
{EngWord} { System.out.println(yytext());}
. { System.out.println(yytext());}
```