

IB047

Unix Text Tools for Corpus Processing

Pavel Rychlý

pary@fi.muni.cz

March 8, 2024

Unix Text Tools Tradition

- Unix has tools for text processing from the very beginning (1970s)
- Small, simple tools, each tool doing only one operation
- Pipe (pipeline): powerful mechanism to combine tools

Short Description of Basic Text Tools

- `cat` concatenate files and print on the standard output
- `head` output the first part (few lines) of files
- `tail` output the last part (few lines) of files
- `sort` sort lines of text files
- `uniq` remove duplicate lines from a sorted file
- `comm` compare two sorted files line by line
- `wc` print the number of newlines, words, and bytes in files
- `cut` remove sections (columns) from each line of files
- `join` join lines of two files on a common field
- `paste` merge lines of files
- `tr` translate or delete characters

Short Description of Basic Text Tools

`egrep` prints lines matching a pattern

`(g)awk` pattern scanning and processing language

`sed` stream editor, use for substring replacement

e.g.: `sed 's/./\u&/g'` – translate to upper case

info run `info` and select from a menu or run directly:

- `info coreutils`
- `info head`, `info sort`, ...
- `info gawk`

man ■ `man 7 regex`
■ `man grep`, `man awk`, `man tail`, ...

-help most tools display a short help message on the
`--help` option

- `sort --help`, `uniq --help`, ...

Unix Text Tools Packages

Where to find it

- set of system tools
- different sets and different features/options on each Unix type
- GNU textutils
- GNU coreutils – textutils + shellutils + fileutils
- other GNU packages: grep, sed, gawk

Unix Text Tools Packages

Where to find it

- set of system tools
- different sets and different features/options on each Unix type
- GNU textutils
- GNU coreutils – textutils + shellutils + fileutils
- other GNU packages: grep, sed, gawk
- installed on all Linux machines
- on Windows: install mingw32/cygwin, then coreutils, grep, ...

Text Tools Usage

- command line tools – enter command in a terminal (console) window
- command name followed by options and arguments
- options start with -
- quote spaces and metacharacters: ' , " , \$
- redirect input and output from/to files using < , >
- use `| less` to only display a result without saving

Text Tools Example 1

task Convert plain text file to a vertical text.

input plain.txt

output plain.vert

solutions

Text Tools Example 1

task Convert plain text file to a vertical text.

input plain.txt

output plain.vert

solutions

```
tr -s ' ' '\n' <plain.txt >plain.vert
```

Text Tools Example 1

task Convert plain text file to a vertical text.

input plain.txt

output plain.vert

solutions

```
tr -s ' ' '\n' <plain.txt >plain.vert
```

```
tr -sc a-zA-Z0-9 '\n' <plain.txt >plain.vert
```

Text Tools Example 1

task Convert plain text file to a vertical text.

input plain.txt

output plain.vert

solutions

```
tr -s ' ' '\n' <plain.txt >plain.vert
```

```
tr -sc a-zA-Z0-9 '\n' <plain.txt >plain.vert
```

```
grep -o '[a-zA-Z0-9]*\|[^a-zA-Z0-9 ]'  
plain.txt >plain.vert
```

Text Tools Example 2

task Create a word list

input vertical text

output list of all unique words with frequencies

solutions

Text Tools Example 2

task Create a word list

input vertical text

output list of all unique words with frequencies

solutions

```
sort plain.vert | uniq -c >dict
```

```
sort plain.vert | uniq -c | sort -rn | head -10
```

Text Tools Example 3

task Corpus/list size
input vertical text/word list
output number of tokens/different words
solutions

Text Tools Example 3

task Corpus/list size

input vertical text/word list

output number of tokens/different words

solutions

```
wc -l plain.vert
```

```
wc -l dict
```

```
grep -c -i '^[a-z0-9]*$' plain.vert
```


Text Tools Example 4

task Create a list of bigrams

input vertical text

output list of bigrams

solution

Text Tools Example 4

task Create a list of bigrams

input vertical text

output list of bigrams

solution

```
tail -n +2 plain.vert |paste plain.vert - \  
                        |sort |uniq -c >bigram
```

Text Tools Example 5

task Filtering
input word list
output selected values from word list
solutions

Text Tools Example 5

task Filtering

input word list

output selected values from word list

solutions

```
grep '^[0-9]*$' dict
```

```
awk '$1 > 100' dict
```

Text Tools Debugging

- data driven programming
- cut the pipeline and display partial results
- try single command with a test input

Text Tools Exercise

task Find all words from a word list differing with
s/z alternation only:
apologize/apologise

Text Tools Exercise

task Find all words from a word list differing with
s/z alternation only:
apologize/apologise

solutions

```
tr s z < dict | sort |uniq -d >szaltern
```

- Find all words from a word list differing with s/z alternation only, and each alternation has higher frequency than 50

Text Tools Exercises

- Find all words from a word list differing with s/z alternation only, and each alternation has higher frequency than 50
- and display their frequencies

Text Tools Exercises

- Find all words from a word list differing with s/z alternation only, and each alternation has higher frequency than 50
- and display their frequencies
- Find all words which occurs in the word list only with capital letter (names).

- XML is a text
 - use same tools (textutils, grep, sort, ...)
- API
 - SAX – Simple API for XML
 - DOM – Document Object Model
- analogy of "text" tools for XML

- Simple API for XML
- event driven computation
- events
 - begin/end of an element
 - element attribute
 - text
- a method/function is called for each event
- minimal resources required

- Document Object Model
- XML document is represented by a tree
- methods for accessing items of a document
- methods for editing (making changes)
- all in main memory
- good for a random access

- set of utilities to query, transform, validate, and edit XML documents
- similar to Unix text tools, works on XML
- XPath for queries
- XML export to PYX (text lines format)

XML processing via JSON

- translate XML to JSON and back (github.com/hay/xml2json)
- use **jq** processing tool: (stedolan.github.io/jq/)

- suite of command-line tools for converting to and working with CSV
- <https://csvkit.rtd.org/>
- csvlook: data periscope
- csvcut: data scalpel

```
csvcut -c county,item_name,quantity data.csv
```


- traditionally for building binary programs from sources
- C, C++, Fortran

- traditionally for building binary programs from sources
- C, C++, Fortran
- aa.h, bb.h, aa.c, bb.c, main.c

- traditionally for building binary programs from sources
- C, C++, Fortran
- aa.h, bb.h, aa.c, bb.c, main.c
- create aa.o, bb.o (binary objects), ab.a (library)
- main (runtime binary)
- handling dependencies

- declaration of dependencies
- specification of rules
 - for concrete target (`main` from `main.o`, `ab.a`)
 - generic (from `*.c` to `*.o`)
 - many defaults

Makefile for data

- it is better to process data in steps
- corpus: html – prevert – vert – annotated
- it could be in one pipeline (at the end)
- but we want to see partial results for debugging during development

Makefile for data

- corpus: html – prevert – vert – annotated
- from html to pre-vertical: html2prevert.py

```
%.prev: %.html  
    html2prevert.py <$< >$@
```

```
%.vert: %.prev  
    tokenize $< >$@
```

```
%.tags: %.vert  
    desamb.sh <$@ >$@
```

Makefile for data

- corpus: html – prevert – vert – annotated
- from html to pre-vertical: html2prevert.py

```
%.prev: %.html
    html2prevert.py -skip-h -m 20 -stopw /nlp/cor... <$< >$@

%.vert: %.prev
    sed -e 's/\([0-9]\)\ -/\1-/g' $< | tokenize |grep -v '^_' >$@

%.tags: %.vert
    desamb-utf8-majka.sh -skipdis <$@ | sed -e 's/^@.*/@\tk4' >$@
```

■ configuration options in variables

```
MAJKA=/nlp/projekty/ajka/bin/majka
%.annot: %.vert
    $(MAJKA) -p <$@ >$<
```

■ list of files/targets

```
PREFS=4 5 6 7 8 9 $(shell seq -w 00 17)
DIRS=$(wildcard SPACE14/20??)
```

```
corps: $(DIRS:%=%.cvert)
```

```
%.cvert: $(PREFS:%=%/\%.vert)
    cat $^ >$@
```

■ variables from commandline: make PREFS='1 2 3'

Make

- run in parallel: `make -j 8`
- run in max load: `make -l [load]`
- dry run: `make -n`
- remake all: `make -B`

- Automation and Make
<https://swcarpentry.github.io/make-novice/>
- Data Science at the Command Line
<https://jeroenjanssens.com/dsatcl/>