Textutils

GNU Text Utilities

Introduction

- This document is derived from GNU Coreutils Manual, version 9.1
  http://www.gnu.org/software/coreutils/manual/coreutils.html
-
-

Introduction to Introduction

- ➔ Why Textutils?
  - ◆ Text streams are universal interfaces.
  - ◆ Solve many common problems.
  - ◆ Don't reinvent the wheel.
  - ◆ Part of computing history.
- ➔ Their limitations led to the creation of modern scripting languages.

Motivation

- ➔ Czech Wikipedia dump (2.7 GB)

```
<page>
  <title>Main Page</title>
  <ns>0</ns>
  <id>48</id>
  <redirect title="Hlavní strana" />
  <revision>
  ...
```

- ➔ Python (1m 25s)

```
import sys
import re

title_pattern = re.compile("^    <title>(?P<title>.*)<\/title>")
redirect_pattern = re.compile("^    <redirect title=\"(?P<redirect>.*)\".*")
title = ""
redirect = ""

for l in sys.stdin:
    result = title_pattern.search(l)
    if result:
        title = result.group("title")
    result = redirect_pattern.search(l)
    if result:
        redirect = result.group("redirect")
        print(title + "\t" + redirect)
```

- ➔ Coreutils (21s)

```
sed -n "/^    <title>/{s/^    <title>//;s/<\/title>//;h}; /^    <redirect
title=/{s/^      <redirect title=\"//;s/\".*//;H;x;s/\n/\t/;p}"
```

Dennis Ritchie (standing) and Ken Thompson begin porting UNIX to the PDP-11 (around 1971)
http://www.catb.org/esr/writings/taoup/html/ch02s01.html

Text Utils Introduction

- Origins in the original AT&T UNIX since the '70
- GNU Coreutils, mostly compatible with POSIX
- Manipulates text files
- Unix Philosophy – **"Do one thing well"**
- Specific tasks done by combining into a pipeline

```
$ cat *.txt | tr ',.' '  ' |
> tr -s ' ' '\n' | sort |
> uniq -c | sort -nr | head -n5
```

Unix Philosophy

- *Mike Gancarz* (one of X Window system designers)
  - Small is beautiful.
  - Make each program do one thing well.
  - Build a prototype as soon as possible.
  - Choose portability over efficiency.
  - Store data in flat text files.
  - Use software leverage to your advantage.
  - Use shell scripts to increase leverage and portability.
  - Avoid captive user interfaces.
  - Make every program a filter.

Text Utils Introduction

- **`foo [option] [file]`**
- in general – input from `stdin`, output to `stdout`
- `foo --help` for common options
- `man 1 foo` for manual page on utility `foo`
- `info foo`

cat: concatenate and write files

- writes concatenation of files to stdout

```
cat 1.txt 2.txt 3.txt > 123.txt
```

- the `'-'` sign for `stdin`:

```
cat head.txt - tail.txt
```

- note: proper text files always end with a newline
- ~~`cat file.txt | grep foo`~~

```
grep foo < file.txt
```

head: output the first part of files

- prints the first part (10 lines default) of each file
- options
  - `-c` *n* print first *n* bytes
  - `-n` *n* print first *n* lines
- if more than 1 input, print a `'==> file name <=='` header for each file, unless `-q` specified
- if *n* starts with `'-'`, print all but *n* last bytes/lines of each file

tail: output the last part of files

- ➔ prints the last part (10 lines default) of each file
- ➔ options
  - `-c` *n* print last *n* bytes
  - `-n` *n* print last *n* lines
- ➔ if more than 1 input, print a `'==> file name <=='` header for each file, unless `-q` specified
- ➔ if *n* starts with `'+'`, start with *n*th byte/line
  - `tail -n+2` skip the first line of input
- ➔ `-f` wait for additional data (monitoring log files)

split: split a file into fixed-sized pieces

➔ **`split [option] [input [prefix]]`**

`-l` *`lines`*	put *lines* lines per output file
`-b` *`bytes`*	put *bytes* bytes per output file
`-d`	use numeric suffixes instead of alphabetic

➔ default prefix `'x'`

`xaa`, `xab`, `xac`, `...`

csplit: split a file into a context-determined pieces

- **`csplit [option] input pattern`**
- patterns define segments
  - `'n'` current line up to line *n*
  - `'/regexp/[offset]'` current line up to the next line of the input file that contains a match for *regexp* *±offset*
  - `'%regexp%[offset]'` ignore segment from current line up to line that contains a match
  - `'{repeat-count}'` repeat the previous pattern repeat-count times, or `*` for infinity

csplit

```
$cat docs.xml
<docs>
<doc>
Lorem ipsum...
</doc>
<doc>
...dolor sit...
</doc>
<doc>
...amet...
</doc>
</docs>
```

```
$cat xx00
<doc>
Lorem ipsum...
</doc>
$cat xx01
<doc>
...dolor sit...
</doc>
$cat xx02
<doc>
...amet...
</doc>
$cat xx03
</docs>
```

```
csplit docs.xml '%^<docs>%1' '/^<\/doc/1' '{*}'
```

wc: print newline, word and byte counts

- **`wc [option] [file]`**
- options
  - `-c` only byte counts
  - `-l` only newline counts
  - `-w` only word counts

sort: sort text files

- **`sort [option] [file]`**
- three modes of operation
  - sort (default)
  - `-c` check for sorted input
  - `-m` merge already sorted files
- sort key selection
  - `-k pos1[,pos2]` select fields for sort field *pos1* to *pos2*
  - `-t separator` use *separator* instead of non-blank to blank transition

sort

- options affecting sort order
  - `-d` dictionary order (ignore non-alphanumeric chars)
  - `-f` fold lowercase to uppercase (ignore case)
  - `-g` general numeric sort (strtod, floating point)
  - `-M` month sort
  - `-n` numeric sort
  - `-r` reverse sort
  - `-b` ignore leading blanks
- environment variables
  - `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`
  - `LC_ALL=C sort` sort by byte values
- other
  - `-s` stable sort

sort environment

```
$cat file.txt
žula
zahrada
chropyně
čekanka
cidr
adresa
Alojs
```

```
$LC_ALL=cs_CZ.ISO-8859-2 sort file.txt
adresa
Alojs
cidr
čekanka
chropyně
zahrada
žula
```

```
$LC_ALL=C sort file.txt
Alojs
adresa
chropyně
cidr
zahrada
žula
čekanka
```

sort keys

```
$cat data.csv
1020,Aglája,Vopajšlíková,BIT 3r
1021,Josef,Vonásek,MGM 2r

sort -t, -k4.5gr,4 -k3,3 -k2,2 data.csv
```

uniq: uniquify files

- **`uniq [option] [input [output]]`**
- writes the unique lines in the given input
- does not detect repeated lines unless they are adjacent
- options
  - `-c` print the number of times each line occurred along with the line
  - `-u` print only the unique lines

```
sort | uniq -c | sort -rn
```

comm: compare two sorted files

- **`comm [option] file1 file2`**
- outputs three columns (separated by TAB)
  - lines unique to *file1*
  - lines unique to *file2*
  - lines common to both files
- options
  - `-1`, `-2`, `-3` suppress printing of column *1, 2, 3*
- `comm -23 foo bar`
  - prints only the lines in *foo* not in *bar*

comm

- ➔ compared files must be sorted according to the same `LC_COLLATE`, `LC_CTYPE`, `LC_ALL`
- ➔ check `sort -c` first
- ➔ some implementations don't collate same as `sort`, `LC_ALL=C` for `sort` and `comm` is a safe bet

cut: print selected parts of lines

- **`cut option [file]`**
- writes selected parts of each line of each input file
- selected input is written in the same order that it is read, and is written exactly once
- options

  `-b` *`byte-list`*
  `-c` *`character-list`*
  `-f` *`field-list`*
  `-d` *`field-separator`*
- lists are sequences of ranges

  ```
  cut -d':' -f1,5-7
  ```

paste: merge lines of files

- ➔ writes lines consisting of sequentially corresponding lines of each given file
- ➔ options
  - `-d` delims, specifies a list of delimiter characters (TAB by default)
  - `-s` paste the lines of one file at a time rather than one line from each file

paste

```
$cat num2
1
2
```

```
$cat let3
a
b
c
```

```
$paste num2 let3
1       a
2       b
        c
```

```
$paste -d',;' -s num2 let3
1,2
a,b;c
```

```
$paste -sd+ num2 | bc
```

join: join lines on the common field

- **`join [option] file1 file2`**
- options

`-1 field`	join on field number field of *file1*
`-2 field`	join on field number field of file2
`-t char`	field separator
`-o list`	output list, `'filenum.fieldnum'`, or `'0'`
`-e string`	replace empty output field with string
`-a1, -a2`	also print unpairable lines from *file1*, *file2*
`-v1, -v2`	like `-a`, but suppress joined output lines

- input files should be pre-sorted on the join field

join

```
$cat scores1
xvopaj00:5
xnovak00:10
xzacha05:20
```

```
$cat scores2
xvopaj00:9
xnovak00:7
xurban04:4
```

```
$sort -t: -k 1,1 scores1 > scores1.sorted
$sort -t: -k 1,1 scores2 > scores2.sorted
$join -t: scores1.sorted scores2.sorted
xnovak00:10:7
xvopaj00:5:9
```

```
$join -t: -v2 scores1.sorted scores2.sorted
xurban04:4
```

```
$join -t: -a2 -o0 -o1.2 -o2.2 -e 0 scores1.sorted
scores2.sorted
xnovak00:10:7
xurban04:0:4
xvopaj00:5:9
```

join

➔ What references non-existing words in the dictionary?

```
$cat dict.sorted
car:Device used for moving people around.
notebook:A portable computer.

$cat refs.sorted
automobile:car
laptop:notebook
snowstorm:blizzard

$join -t: -1 1 -2 2 -v 2 -o2.1 dict.sorted refs.sorted
snowstorm
```

join

➔ Print the most frequent word forms, whose lemmata are not included in the dictionary.

`frequent_forms.txt`	word forms sorted according to their frequency
`lemmata.txt`	form:lemma
`dictionary.txt`	lemma:sense

```
$cat -n frequent_forms.txt | tr '\t' ':' |
> sort -t':' -k2,2 |
> join -t':' -1 2 -2 1 - lemmata.txt.sorted | sort -t':' -k3,3 |
> join -v1 -t':' -1 3 -o1.2 -o1.1 - dictionary.txt.sorted |
> sort -n -t':' -k1,1 | cut -d: -f2
```

tr: translate, squeeze, and/or delete characters

- → `tr [option] set1 [set2]`
- → copies input to output, performing one of the following operations:
  - translate, and optionally squeeze repeated characters in the result
  - squeeze repeated characters
  - delete characters
  - delete characters, then squeeze repeated characters from the result
- → very fast

tr, sets

- options
  - `-c` replace *set1* with complement
  - `-s` squeeze
- sets
  - `\n`, `\t`, `...` special characters (newline, tab, ...)
  - `\ooo` octal ASCII value
  - ranges
    - `m-n` such as `'0-9'`
    - `[c*n]` *n* copies of character *c*
    - `[c*]` fill the *set2* with *c* to length of *set1*
    - `[:class:]` alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit

tr: translating

➔ examples

`tr yz zy` translate *y* to *z* and *z* to *y*

`tr a-z A-Z` uppercase

```
tr '[:lower:]' '[:upper:]'
tr -sc '[:alnum:]' '[\n*]'
```

replace every non-alpha-numeric character with a newline

`-c` negates the `[:alnum:]`

`-s` (squeeze) removes repeated characters from *set2*

tr: delete and squeeze

- `tr -d '0-9'` delete all numbers
- `tr -s '\n'` convert each sequence of newlines to a single newline

grep: print lines matching a pattern

- **`grep [options] pattern [file]`**
- options
  - `-f` obtain patterns from file, one per line
  - `-v` invert the sense of matching, to select non-matching lines
  - `-i` ignore case distinctions in both the *pattern* and the input *files*
  - `-F` interpret *pattern* as a list of fixed strings, separated by newlines, any of which is to be matched

- *pattern* may be basic (`-G`), extended (`-E`), or Perl (`-P`) regular expression

grep extended RE

- characters
  - most characters, numbers are regular expressions (matches the character itself)
  - `.` matches any character
- if `A` and `B` are RE,
  - `AB` matches concatenation if `A` and `B`
  - `A|B` matches any of `A` or `B`
- repetition operators matches preceding items
  - `?` at most once
  - `*` zero or more times
  - `+` one or more times
  - `{n}` *n* times
  - `{n,}` *n* or more times
  - `{n,m}` at least *n* but no more than *m*

grep character class

- character classes
  - `[0123]` set of characters 0123
  - `[0-3]` set of characters 0123
  - `[^0-9]` any not-a-number character
  - `[[:class:]]` alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit
- backslash characters
  - `\b` empty string at the edge of a word
  - `\B` empty string not at the edge of a word
- anchors
  - `^` matches empty string at the beginning of a line
  - `$` matches empty string at the end of a line

grep

```
➔ grep '\brat\b'
➔ grep -E '\.{3}$'
➔ grep -iE '\b(hello|hi|cheers)\b'
```

matches a line *'I smell a rat'*,
but not *'ratification'*

lines ending with . . .

grep

- ➔ in basic regular expressions the metacharacters `?`, `+`, `{`, `}`, `|`, `(`, and `)` lose their special meaning; instead use the backslashed versions `\?`, `\+`, `\{`, `\}`, `\|`, `\(`, and `\)`
- ➔ more about regular expressions in another ISJ lecture

"You've got a problem, and you've decided to use regular expressions to solve it.

Ok, now you've got two problems..."

sed: the streaming editor

- **`sed [options] [script] [file]`**
- modifies the stream
- the sed script language is Turing complete
- options

  `-f file` load script from a *file*

sed: substitution

- **`s/regexp/replacement/flags`**

- `g` flag applies the substitution to all matches
- `\N` references the Nth `\(` and matching `\)`
- the `/` can actually be any other character

```
sed 's/<NAME\/>/John/g'
sed 's/\([^:]*\).*/\1/' /etc/passwd
echo "Hello World!" | sed 's/\(\b[A-Z]\)/\(\1\)/g'
sed 's!href="http://example.com/\([^"]*\)"!href="./\1"!g'

sed settee <<< bottom      boom
sed serene <<< wire        wine
sed stoutest <<< you       yes
sed statement <<< dated    demented
```

AWK

- programming language for text processing
- predecessor of Perl
- searches file for patterns, do some action on matched line

  ```
  pattern {awk-commands}
  ```

- each line is a set of fields

```
awk 'BEGIN {print("Hello World!")}'
awk -F":" '{print $1 }' /etc/passwd
awk 'length($0) > 72' text.txt
awk '{print($3, $2, $1)}'
awk 'BEGIN {FS=":"; OFS=":"} {print($1, $2+$3)}' <
scores.txt
```

Makefile

- ➔ detects which files (from a large project) have been changed since the last time the program was compiled and recompiles them
- ➔ Makefile is a set of targets, prerequisites and commands
- ➔ documents very well how each file was created

```
CATS = C1 C2 C3 C4 C5 C6

EXTS = freq freq.srt.missing

ALLFILES = $(foreach s, $(EXTS), $(CATS:=.$(s)))

all: $(ALLFILES)

clean:
    rm -f $(ALLFILES)

# number of occurrences of words that don't contain a capital
letter (single occurrences are not included)
%.freq: %
    grep -v "[[:upper:]]" $< | sort | uniq -c \
        | sort -rn | awk '{if ($$1 > 1) print;}' > $@
%.srt: %
    sort -k2,2 $< > $@

# words not included in the dictionary
%.missing: % dictionary
    join -1 1 -v 1 $^ > $@
```

Makefile

- commands begin with **a tab**
- commands are executed by making a new subshell for each line
- if you want to use a command to affect the next command, put the two on a single line with a semicolon between them

```
mytarget:
    cd mydirectory; touch myfile
```

- be aware of implicit rules (and file extensions)
- `$(var)` returns a value of variable `var`, write `$` instead of `$$` in nested scripts
- automatic variables
  - `$@` the file name of the target of the rule
  - `$<` the name of the first prerequisite
  - `$?` the names of all the prerequisites that are newer than the target
  - `$^` the names of all the prerequisites

Makefile

- options
  - `-n` print the recipe that would be executed, but do not execute it
  - `-p` print the data base (rules and variable values) that results from reading the makefiles, then execute as usual

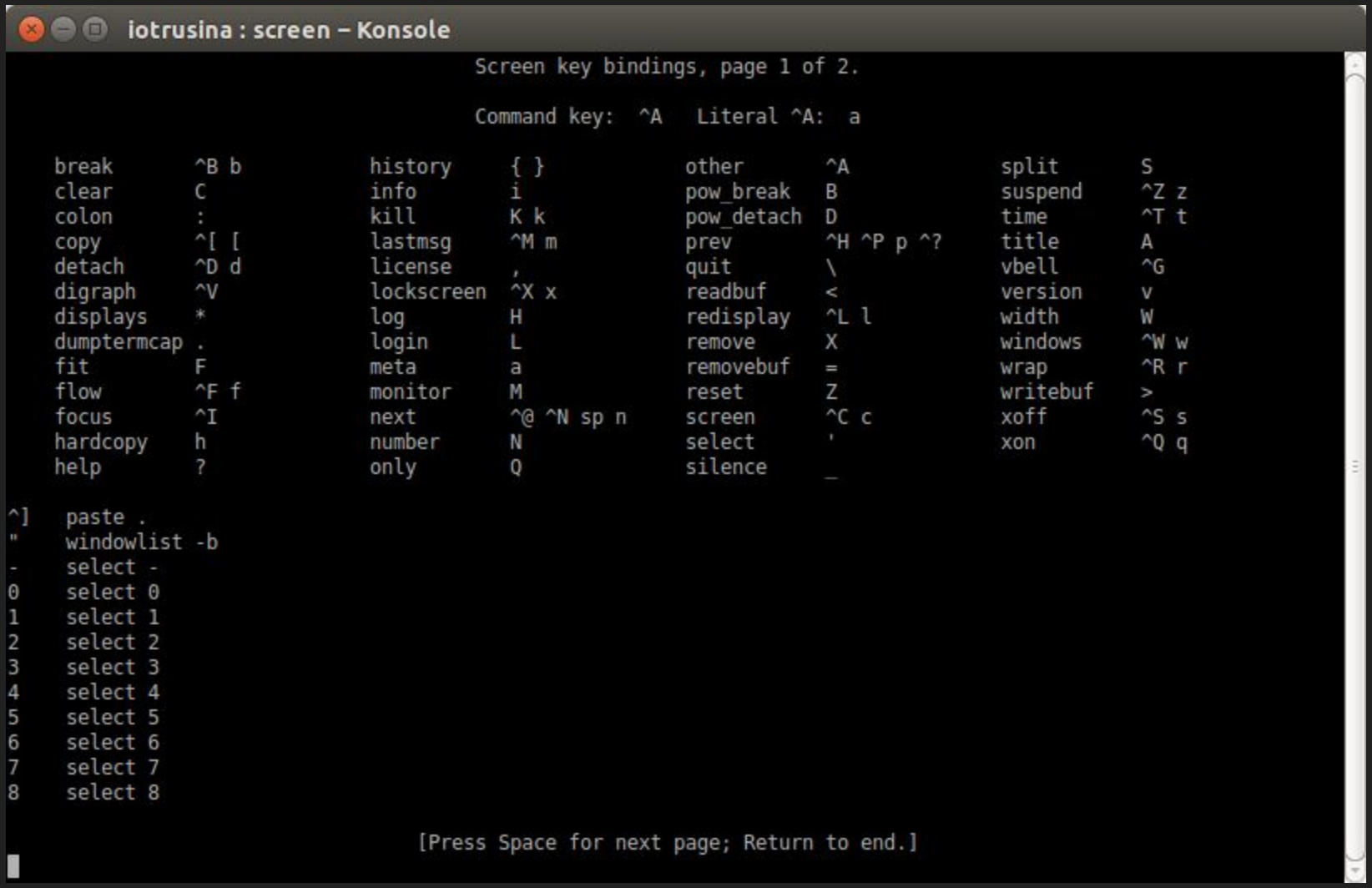
Bonus: How To Use Screen

- ➔ Use multiple shell windows from a single SSH session.
- ➔ Keep a shell active even through network disruptions.
- ➔ Disconnect and reconnect to a shell sessions from multiple locations.
- ➔ Run a long running process without maintaining an active shell session.

Bonus: How To Use Screen

- ➔ creates a named session
  ```
  screen -S sessionname
  ```
- ➔ detaches screen session
  ```
  CTRL+A, D
  ```
- ➔ resumes a detached screen session
  ```
  screen -r sessionname
  ```
- ➔ attempts to resume the youngest detached screen session it finds
  ```
  screen -R sessionname
  ```
- ➔ prints a list of pid.tty.host strings and creation timestamps identifying your screen sessions
  ```
  screen -list
  ```

Bonus: How To Use Screen



Conclusions

- ➔ We have a general idea about Textutils.
- ➔ We know that grep, sed and awk exist.
- ➔ We know how to use Makefile to manage scripts.
- ➔ We will now better understand modern script languages.

References

- GNU Coreutils manual http://www.gnu.org/software/coreutils/manual/coreutils.html
- David MacKenzie et al., GNU Textutils, Free Software Foundation, 1997
- The UNIX School http://www.theunixschool.com/
- Online regex tester and debugger https://regex101.com/
- Regex Crossword https://regexcrossword.com/