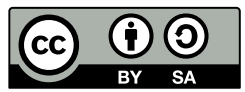
Filters, microlanguages and Shell scripts

Saulius Gražulis

2009 ruduo

Vilnius University, Faculty of Mathematic and Informatics
Institute of Informatics



Filters

- Filter – a program which reads its standard input and writes its output to standard output
- Command `cat` can act as a filter:
  - `sh> echo "I Dalis" | cat - I-d.txt pabaiga.txt > pirma-dalis.txt`

Get Regular ExPression

grep

- Usage: grep [OPTION]... PATTERN [FILE]...
- command `grep` searches for lines matching regular expression PATTERN in its standard input or files supplied as its arguments.
- E.g.:
  - `sh> grep root /etc/passwd`
  - `root:x:0:0:root:/root:/bin/bash`
  - `sh> cat 1.dat 2.dat 3.dat | grep AVERAGE`
  - `...`
- `grep` patters usually contain shell metasymbols thus it is recommended to provide patterns in quotes.

`grep` regular expressions

- Ordinary symbols (letters, digits) match themselves
- Decimal dot (“.”) – matches any single symbol
- Symbols or their ranges in brackets (“[]”) – match any symbol from the given set
- Asterisk (“*”) – modifies preceding regular expression to match it 0 or more times

grep regular expressions

- Question mark (“?”) – modifies preceding regular expression to match it zero (0) or one (1) time
- Caret (“^”) – matches the beginning of the line
- Doller sign (“$”) – matches the end of the line
- Backslash (“\”) – makes the following symbol lose its special meaning

grep usage examples

- `sh> grep 'Part I' book.txt`
- `sh> grep 'ATOM ' 1knv.pdb`
- `sh> grep '^ATOM' 1knv.pdb`
- `sh> grep '^ATOM ' 1knv.pdb`
- `sh> grep '^SCALE[123]' 1knv.pdb`
- `sh> grep '^SCALE[1-9]' 1knv.pdb`

Handling PDB files with *x commands – examples

Each line of a PDB file (“record”) starts with 6-symbol keyword; data values are located in fixed columns of a record (e.g. name of a chain is provided in 22 position of ATOM record).

- Selects only ATOM records from a PDB file:

```
sh> grep '^ATOM␣␣' 1knv.pdb
sh> grep "^ATOM␣␣" 1knv.pdb
```

- Selects only CRYST, ATOM, HETATM and END records from a PDB file (-E argument stands for “use extended regular expression”):

```
sh> grep -E '^(CRYST1|ATOM␣␣|HETATM|END)' \ 1knv.pdb
```

- Lists chains in a PDB file:

```
sh> grep -E '^(ATOM␣␣|HETATM)' 1knv.pdb \
| cut -b 22-22 | sort | uniq
```

sed microlanguage

- Usage: sed [OPTION]... script [input-file]...
- `sh> echo -e "vienas\ndu\ntrys" > tekstas.txt`
- `sh> cat tekstas.txt`

```
vienas
du
trys
```

- `sh> sed -e 's/vienas/1/' tekstas.txt`

```
1
du
trys
```

- `sh> sed -e 's/d./2/' tekstas.txt`

```
vienas
2
trys
```

awk programming language

Aho, Weinberger, Kernighan

Program awk allows, as does grep, select lines of a file by matching them using regular expressions (written between slashes, /.../). Moreover, each matching line can be processed using tiny program (written in curly braces, {...}).

- Usage: awk [POSIX or GNU style options] -f progfile [–] file ...
  Usage: awk [POSIX or GNU style options] [–] 'program' file ...
- `sh> awk '/PATTERN/ { print }' book.txt`
- `sh> awk '/^XYZ/ { if( $1 > 0 ) print }' coord.dat`

Other prominent *x filters

GNU systems (like Linux) have many useful filters; some of them are listed below. For more information see `'info coreutils'` (GNU `coreutils` package description).

- `tr` – "translate" – replaces requested symbols by other symbols, or removes them altogether (with -d option):
  - `sh> tr "\r" "\n" < book.mac-txt > book.linux-txt`
  - `sh> tr -d "\r" < book.dos-txt > book.linux-txt`
- `wc` – "word count" – counts lines, words and symbols in given files. Counts only lines (with -l option), only words (with -w option) or only symbols (with -c option):
  - `sh> wc book.txt`
  - `sh> wc -w < book.txt`
  - Counts .txt files in working directory:
    `sh> ls *.txt | wc -l`
  - Counts atoms in PDB files:
    `sh> grep '^ATOM ' *.pdb | wc -l`

Perl as command line filter

Perl programming language can be used to write short scripts-filters in the command line:

- Finding the ASCII code of a symbol:
  - `sh> perl -e 'printf "%d\n", ord("A")'`
  - `sh> perl -e 'printf "0x%02X\n", ord("A")'`
- Modify each line with regular expression and print it (`sed` analogue):
  - `sh> perl -pe 's/mine/yours/g' *.txt`
- Modify each line with a Perl program; modified line is printed afterwards (`awk` analogue):
  - `sh> perl -ne 'print unless /^\s*$/' *.txt`

An example of more complex task

Find *anagrams* (words produced by permutations of the same letters) in text files. E.g. words “lime” and “mile” are anagrams.

Producing all the anagrams would be impractical ($\sim n!$ combinations for each word of length n)

Solution: sort (order) letters of words and use the generated lines as keys to identify anagrams.

A program for anagram search

- anagrams.sh:

```
#!/bin/sh
# Find all words that are anagrams in input files
cat $* \
| tr "\r\t" "␣" \
| perl -040 -l012 -ne 'print' \
| perl -CS -lne 'print join("",sort(split(""))), "␣", $_' \
| sort -k1 | uniq \
| perl -lane \
    'sub print_anagrams(@) {
        if( @_ > 1 ) {
            for( @_ ) { print $_->[1] }; print ""
        }
    }
    if( !@p || $p[0][0] eq $F[0] ) {
        unshift(@p,[@F])
    } else {
        print_anagrams( @p );
        @p=([@F])
    }
    END {
        print_anagrams( @p );
    }'
```