CS3281
UNIX Programming

Part 8

Awk Programming

Zoltan KATO (cs3281@comp.nus.edu.sg)
S16 #06–12

Getting Started

- **awk** is a pattern scanning and text processing language.
  - it is useful to generate reports, summarize large amount of data or filter text
  - An awk program consists of a series of **rules**:

    ***pattern* { *action* }**
    - a **comment** starts with a **#** and continues to the end of the line
  - The input is split into lines and each matching rule is applied to the line in the order they appear in the program file.
- running awk programs:
  - **awk *'program'* *file1 file2* ...**
  - **awk -f *program-file* *file1 file2* ...**
    - make use of interpreter files:

      #!/usr/bin/awk -f
  - if no input file is specified then the program is applied to **stdin**
  - it produces its output on **stdout**

Patterns

- Patterns using regular expressions
  - ***/regexp/*** : tested against an entire input line
  - ***exp ~/regexp/*** : true if ***exp*** (as a string) is matched by ***regexp***
  - ***exp !~/regexp/***: true if ***exp*** is not matched by ***regexp***
- Arithmetic and character relational operators:
  - **< <= == >= >**
- Patterns may be combined using the logical operators: **| |** (OR) **&&** (AND). Patterns may be grouped using: **( )**
- ***/p1/ , /p2/***: range pattern matches all input lines between a line matching ***p1*** and a line matching ***p2***, inclusive.
- Special patterns:
  - **BEGIN** matches the beginning of file (before processing starts)
  - **END** matches the end of file (after processing is finished)
- Missing pattern matches every line

Regular Expressions

c	matches the character **c**
.	matches any character, including *newline*
^	matches the beginning of a string
$	matches the end of a string
[*abc*]	matches any one of the enclosed characters (***a***, ***b***, or ***c***)
[^*abc*]	matches any character, except the enclosed ones
r1*|*r2	matches either ***r1*** or ***r2*** (alternation)
r1r2	matches ***r1***, and then ***r2*** (concatenation)
***r*+**	matches one or more ***r***
r*	matches zero or more ***r***
***r*?**	matches zero or one ***r***
(*r*)	grouping (matches ***r***)

/abc/ - matches any line containing the string **abc** *anywhere*

/abc$/ - matches any line ending with **abc**

/wh+y/ - matches any line containing **why**, **whhy**, **whhhy**, ...

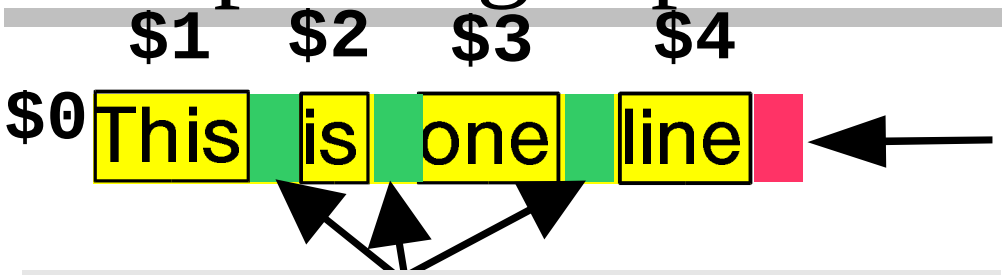
Actions

- A missing action is equivalent to **`{ print }`**
- Operators are the same as C operators:
  - **`+ - * / % ++ -- += *= =`** ... **`^`** (exponentiation)
  - ***space*** − string concatenation, **`$`** − field reference
- Control statements (syntax similar to C statements):
  - **`if (`*condition*`)`** ***statement*** **`else`** ***statement***
  - **`while (`*condition*`)`** ***statement***
  - **`for (`*expr1*`;`** ***expr2*`;`** ***expr3*`)`** ***statement***
  - **`{`** ***statements*** **`}`**
- I/O statements. Output can be redirected using **`>`**, **`>>`**:
  - **`print`** ***expr−list*** : prints expressions, current line if no expr.
  - **`printf`** ***fmt expr−list*** : formatted printing (***fmt*** is like in C)
  - **`next`**: stops current processing, reads next line and starts over
  - **`getline`** ***var*** : set ***var*** from next input line

Variables

- No need to declare variables, just use them:
  - `n = 5  str="string"  newline="\n"`
  - a value is either numeric or string (conversion is done by **awk**)
  - string constants are enclosed in double quotes: `"str 1\n"`
  - uninitialized variables are set to `""` (=`0` when used as number)
- Built−in variables set by **awk**, they can be changed from awk programs:
  - `NR` − line number of current line (=number of lines seen so far)
  - `NF` − number of fields in the current line
  - `$1`, `$2`, ..., `$NF` − fields in the current line (`$n` is allowed where `n` is a numeric variable). `$0` − the entire line
  - `FS` − input field separator (default: space or tab)
  - `RS` − input line (record) separator (default: newline)
  - `FILENAME` − current input file name

Splitting Input into Lines and Fields



FS character: one or more space, tab by default. Leading and trailing whitespace are ignored

FS="" Each individual character becomes a separate field

FS="c" means fields are separated by a single character **c**

RS character (newline by default)

RS="" means lines (records) are separated by one or more blank lines (useful for multi-line records) newline is always a field separator in such cases!

RS="c" means lines are separated by a single character **c**

One liners:

NF>0 : deletes blank lines

length($0)>80 : print lines longer than 80 characters

{print NR, $0} : print line numbers

NF>1 {t=$1; $1=$2; $2=t; print} : swap first two fields

END { print NR } : counts lines of the input file

```
NR%2 == 0 {print >> even_lines}
NR%2 == 1 {print >> odd_lines}
```

Built−in Functions

- String manipulation:
  - **`tolower(`*str*`)`** **`toupper(`*str*`)`** − converts ***str*** to lower (resp. upper) case and returns the result.
  - **`sprintf(`*fmt*`,` *expr−list*`)`** − like **`sprintf()`** in C.
  - **`length(`*str*`)`** − returns the length of the string ***str***
  - **`match(`*str*`,` *regexp*`)`** − returns the position in ***str*** where the regular expression ***regexp*** occurs, or 0 if it is not present.
  - **`gsub(`*regexp*`,` *repl*`,` *str*`)`** − for each substring matching the regular expression ***regexp*** in ***str***, substitute ***repl*** and return the number of substitutions.
  - **`gensub(`*regexp*`,` *repl*`,` *n*`,` *str*`)`** − Replace the ***n***'th occurrence and return the changed string. ***str*** is not modified!
- System functions:
  - **`system(`*cmd*`)`** − execute ***cmd*** and return the exit status.
  - **`strftime()`** − returns the current date and time as a string