awk

Named from its authors: Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan

Def: A powerful programming language disguised as a utility

Functionality

- awk reads the input file line by line and performs actions
  - Prints only when specified
- awk has two UNIX options
  - "-F" option: specifies the input field separator (more rare)
  - "-f" option: names the script file
  - If script is on command line, placed in single quotes
  - if there is no input file, the input comes from the keyboard which is designated by "-"
- Execution- awk requires one or more instruction
  - On the command line
    - Ex: `awk 'pattern {action}' input_file`
  - awk script
    - Scripts are suffixed with .awk
    - Executed using '-f'
    - Ex: `awk -f scriptFile.awk input_file`

Fields and Records

- Files are a collection of fields and records
- Fields- a units of data that have informational content
  - Ex: "ls" outputs fields ranging from the permissions to the filename
  - Fields are each separated by white space
- Record- a collection of fields treated as a unit
  - All data is related
- Files containing records are called data files, "fileName.dat"
- awk uses data files as input, but text files can be used
  - Lines of the text become records, just of varying numbers of fields

Buffers and Variables:

- Buffer- area of memory which holds data while processing occurs
  - Field buffers: represented by $1,$2,…,$n where n is the number of fields in input file
  - Record buffer: there is only one, $0
    - The concatenation of all the field buffers separated by a field separater
- Variables- system and user-defined
  - 12 system variables: 4 controlled by awk, 8 have defaults be can be changed
    - FS, RS, OFS, ORS, NF, NR, FNR are the most common
  - User-defined: not declared, come into existence first time referenced

Script

- awk scripts are the instructions containing three parts
  - Begin: designated by 'BEGIN', followed by instructions enclosed in a set of braces
    - Initialize variables, create report headings, and other processing necessary before file processing
  - Body: a loop that processes the records of a file one at a time
    - The loop is contained in one or more set of braces
  - End: designated by 'END', occurs after all input data has been read
    - Accumulated information can be analyzed and printed

Example script file:

```
# Begin Processing
BEGIN {print "Print Totals"}

# Body Processing
{total = $1 + $2 + $3}
{print $1 " + " $2 " + " $3 " = "total}

# End Processing
END {print "End Totals"}
```

Patterns

- Identifies which records receive actions- if pattern returns true (matches) action takes place
- Statement w/o a pattern is always true
- Simple Patterns-
  - Begin and End (already covered)
  - Expressions- 4 types
    - Regular- those covered in chapter 9 and 10
    - Arithmetic- match when nonzero (-, +, ++, =, etc.)
    - Relational- can be string or algebraic compares (<, >, ==, etc)
    - Logical- operators to combine two or more expressions (&&, ||, !)
- Range Patterns-
  - Associate with a range of records, there are two simple patterns
  - Starts with first record to match the first pattern and ends with next record to match the second pattern
  - Ex. `awk 'NR == 8, NR == 13 {print NR, $0)' TheRaven`

Actions

- Instructions or statements, they act when pattern is true
- One-to-one relationship between action and pattern
- Action must be in braces
- Block- set of braces containing pattern and action
  - Considered one statement
  - Nested block- a block inside a block
- Statements of an action must be separated with: new line, semicolon or set of braces
- 5 different types of statements
  - Expressions- ex. {total += ($3 + 9)}
  - Output- 3 types: print, printf, and sprintf
    - Print- writes specified to standard output file, must be separated with commas, when nothing specified entire record is printed
    - Printf- a formatting print with a format string
    - Sprintf- a formatted print that combines two or more fields into a string
  - Decision- a typical if-else statement
  - Loop- typical loops: while, for, and do-while
  - Control- there are 3: next, get a line, and exit

Print and Control

- Print: ex. `{print $1, $2, $3}`
- Control
  - 'next': terminates processing of current record and pushes to the next
  - 'getline': a function used as a statement
    - Unlike next it continues executing on the next record instead of terminating
    - Input directed to $0 or another variable
    - Returns 1 (success), 0 (end of file), or -1 (read error)
    - Uses redirection operator (<) to get input from another file
  - 'exit': send to the end statement, used for error conditions
- Example:

```
# exchange.awk scrip
#exchanges line 2 by 2
{
if ((getline evenLine) == 1)
  {
    print evenLine
    print $0
  }
 else
   print $0
}
```

Associative Arrays

- Like any other array, but the indexes are represented by strings
- The index is some how associated w/ the element (hence the name)
- There is no ordering imposed
- The index cannot be sorted
- Processing:
  - For…in loop: `for(index_variable in array_name`
  - Creating: ex. `name[$3]`
  - 'delete': deletes an element from the array
    - `Delete array_name[index]`
  - Since indexes are not sorted, so printing associative arrays occurs in no particular order
- Example: `awk –f salesDeptLoop.awk sales1.dat`

String Functions

- awk has a vast number of string functions
- 'length(string)': returns number of characters including whitespace
- 'index(string, substring)': returns the first position of substring in string
  - `index(joshua,ua)` returns 5
- Substring- 2 formats
  - 'substr(string, position)'- returns the substring starting at the desired position
  - 'substr(string, position, length)'- returns substring at the position with the designated length
- Spit- 2 formats
  - 'split(string, array)'- splits the fields of a string by the FS and places them into the designated array (numbered indexes)
  - 'split(string, array, field_separator)'- designates the field separator

String Functions

- Substitution
  - 'sub(regexp, replacement_string, in_string)'- returns true if successful
  - 'gsub'- same format, but a global substitution
- 'match(string, regexp)'- returns true if successful
  - Creates RSTART (position of match) and RLENGTH (length of match)
- 'toupper' and 'tolower'- string parameters and turns lower case to upper and vice versa

Mathematical Functions

- int- truncates floating-point
- rand()- returns next random number
- srand(seed)- seeds random number series
- cos(x), exp(x), log(x), sin(x), sqrt(x)
- Atang2(y,x)- returns arc tangent y/x in range of –pi to pi

User-Defined Functions

- Format

```
function name(parameters)
{
    code
}
```

- No space between name and parameters in function call or definition
- No semicolon need
- No declaring, like all awk

System Commands

- Pipes- can give the date, inside a loop can output users on the system
  - `awk -f date.awk`
- 'system(string)'- checks if the command in the string is successful
  - Returns 0 if successful and 1 if not

Application

- awk has a vast number of applications
- Two examples-
  - Count words and line of a file
    - `awk –f wordCount.awk sonnet.dat`
  - Return just phone numbers
    - `awk –f phones.awk phones.dat`

Sed and Grep

- awk has limitations when it comes to sed, but can perform most of the same actions
- awk can take the place of grep, but it is much slower and less efficient