Lecture #13 – AWK Part I (Chapter 6)

- Background

  AWK is a pattern-scanning and processing language.
  Named after authors (Alfred Aho, Peter Weinberger, Brian Kernighan).
  Design to be powerful string processing utility that is easy to use
  Slower than some alternatives like compiled languages

  Note: Solaris has 2 versions awk & nawk (nawk is the "new" awk)
  Note: gawk is the GNU version of awk

- Processing Description

  An awk program is a text file that consists of one or more statements of the form:

  pattern { action }

  The pattern selects lines from the input file, and awk performs the action on all lines that match the pattern.

  Processing of an input file is as follows:

  1. awk compares the first line in the input file with each pattern in the program file
  2. If a pattern selects the line, then awk takes the action associated with the pattern.
  3. If there are several patterns that match a line, they are executed in the order they reside in the program.
  4. When awk has completed comparisons and actions for the first input line, it repeats the process for the next input line.

- Command line args

  awk [-Fc] –f program file [file-list]
  awk program [file-list]

-f program-file	causes awk to read its program from the given file
-Fc	causes "c" to become the field separator (normally whites pace)

  Examples:

  awk '/^#/ { print $1 }' input_file
  awk –F: -f passwd.awk /etc/passwd
  ls –ls | awk –f richj.awk

- Patterns

  Can be a regular expression enclosed in forward slashes (i.e. /richj/)
  ~ operator tests to see if a field or variable matches a regular expression ($1 ~ /richj/)
  ! operator tests for no match (! /richj/)
  Can use relational operators (<, <=, ==, !=, >=, >)
  Combine expressions with OR (||) and AND (&&)

  There are 2 special patterns BEGIN and END.

  Examples:

  # Print 1st field for all lines that start with capital letter
  /^[A-Z]/ { print $1 }

  # before processing input, initialize some variables
  BEGIN { sum = 0; first = "yes" }

  # print a summary after processing input
  END { print "The sum is ", sum }

- Actions

  The default action is { printf }
  If there are several actions, they can be separated by semicolons ";"

- Comments

  Comments are anything on a line following a pound sign (#)

- Variables

  User variables are declared on use. (DO NOT reference with $var)

  Special Variables:

NR	record number of current record
$0	entire current input line
NF	number of fields in current record
$1 - $n	Fields in the current record
FS	input field separator
OFS	output field separator
RS	input record separator (usually newline)
ORS	output record separator (usually newline)
FILENAME	name of the current input file

Examples:

```
# Print each line from the input file (i.e. similar to cat)
$ awk '{print $0}' employees
$ awk '{print}' employees

# Print each line from the input file with its line number
$ awk '{ print NR, $0 }' infile

# Print each line from the input file with the number of fields on the line
$ awk '{ print $0, NF }' infile

# Change filed separator to colon
$ awk –F: '/Tom Jones/ { print $1, $2 }' infile
$ awk 'BEGIN {FS=":"} /Tom Jones/ { print $1, $2 }' infile
```

- Matching the entire line

  Standalone regular expressions as patterns match against the entire input line.

```
# Print all lines that start with Mary (i.e. similar to grep)
$ awk '/^Mary/' infile

$ awk '/^[A-Z][a-z]+ /' infile
```

- The match operator

  The match operator (~) is used to match an expression within a record or field.

```
# Print all lines where the 1st field is Bill or bill
$ awk '$1 ~ /[Bb]ill/' employees

# Print all lines where the 1st field does not end with "ly"
$ awk '$1 !~ /ly$/' employees
```

- Awk commands in a script file

```
$ cat info

/Tom/ { print "Tom's birthday is ", $3 }
/Mary/ { print NR, $0 }
/^Sally/ { print "Hi Sally." }

$ awk –F: -f info infile2
```

- Comparisons in patterns

```
#  Print all lines where the 3rd field is 5346
$  awk '$3 == 5346' infile

#  Print the 1st field from each line where the 3rd field is bigger than 5000
$  awk '$3 > 5000 { print $1 }' infile
```

- Computation in patterns

```
#  Print all lines where the 3rd field multiplied by the 4th field is bigger than 500
$  awk '$3 * $4 > 500' filename
```

- Logical operators in patterns

```
#  Print all lines where the 2nd field is between 5 and 15
$  awk '$2 >= 5 && $2 <= 15' filename

#  Print all line where the 3rd field is 100 or the 4th field is bigger than 50
$  awk '$3 == 100 || $4 > 50' filename

#  Example of logical not
$  awk '! ($2 < 100 && $3 < 20)' filename
```

- BEGIN and END patterns

```
$  cat summary

BEGIN {
        yearsum = 0;   costsum = 0
        newcostsum = 0;  newcount = 0
}

{  yearsum += $3;   costsum += $5 }

$3 > 90 { newcostsum += $5;  newcount++ }

END {
        printf "Average age of cars is %3.1f years\n", \
                90 – (yearsum / NR)

        printf  "average cost of newer cars is $%7.2f\n", \
                newcostsum / newcount
}
```

```
$  cat find_uid                          find max UID + 1

BEGIN {
        FS = ":"
        saveit = 0
}

$3 > saveit { saveit = $3 }

END { print "Next available UID is " saveit + 1 }

$  cat manuf                             uses associate arrays

{  manuf[$1] ++ }

END { for (name in manuf) print name, manuf[name] }
```