Scripting with Awk

~/IFT383/mod-3

First: some review...

Objectives

- Split lines into fields and format output using Awk
- Use comparison operators in Awk
- Perform pattern matching with ~ and !~ comparators
- pre and post-processing with BEGIN and END keywords
- Use variables in Awk

Awk

- Unlike most UNIX tools; that do one specific thing extremely well. Awk does many things and is sometimes called the “swiss army knife” of UNIX tools
- Awk uses a record and field model to address data within a file
  - records are (by default) are interpreted as lines in a file
  - fields *by default(are seperated by spaces or tabs
- Excels at performing aggregate functions across records; such as calculating the sum or average of a field across a file or subset of records

Awk Syntax

- Similar to **sed**, awk has the general format;
  - awk [iotuins] ‘statement’ [input file]
- Awk can read input from stdin, a file or the terminal
- Awk statement can take any of these firms;
  - pattern {action}
    - Match the provided pattern and perform the corresponding action whenever the pattern matches
  - pattern
    - Print all records (lines) that match the pattern - similar to grep
  - {action}
    - Perform the action to all records in the file

Basic Awk Options and Commands

- Options
  - -F “?”
    - Specify a field separator, where ? is a character or escape sequence
    - Example; awk -F “\f” ‘expression’ fileName
  - -f scriptFile
    - read awk statements from a file, rather than the command line
- commands
  - print
    - similar to “echo” program
  - printf
    - print formatted output
    - example; awk ‘{printf “%s\n%d\n”, $1, $3}’
      - %s is a string, %d is a digit

Accessing fields in a record

- Awk assigns the following variables when it reads a record;
    - $0 the entire record
    - $1, $2 … $n the fields in the record
        - fields are determined for each record; using the field seperator
            - If not specified, the default field separator is space and tab

Field example

```
$cate data
This is the first line
this is the second line
this is the third line

$ cat data | awk ‘{print $4}’
first
second
third
```

Pre and Post-processors

- The **BEGON** pattern can be specified to instruct awk to perform a preprocessor operation before evaluating the records from input
- The END keyword works the same way; performing an action afer all records have been processed

Example coming up next…

BEGIN/END example

```
$ cat buildings
ode    Name            Floors
PCHO    Picacho Hall            3
PRLTA   Peralta Hall           3
AGBC    Agribusiness Cntr               1
TECH    Technology Cntr         1

$ cat ./buildings | awk -F"," 'BEGIN {print "Code\tName\t\tFloors"};
{printf "%s\t%s\t\t%d\n",$1,$2,$3};END {print "All Done!\n"}'

Code    Name            Floors
PCHO    Picacho Hall            3
PRLTA   Peralta Hall           3
AGBC    Agribusiness Cntr               1
TECH    Technology Cntr         1
All Done!
```

Comparison operators

- In the **pattern** portion of out awk statements, we are not limited to writing patterns that are matched against an entire record (line)
- Performing a RegEx match against a specific field using the “~” operator
  - Example; field 2 starts with 100
    - awk ‘$2~/^100/ {print “$0”}’
  - Alternatively; you can invert the match with “!~”
- Match identity of a field using “==”
  - Example; field 3 is exactly “Arizona”
    - awk ‘$3==”Arizona” {print “$0”}’
  - Invert the match with “~=”

Now,
To get complicated...

Objectives

- Use advanced properties of Awk variables and expressions
- Perform calculations using floating point numbers
- Use arrays
- Use conditionals and loops

Awk scripting

- Awk is a full-fledged scripting language
    - Awk scripts by convention, end in the extension **.awk**
    - Awk scripting predates Perl (module 4)
    - Originally developed in 1997, original Awk has been largely superceeded by GNU Awk (gawk)
- Specify an awk script file with the -f option
    - awk -f myScript.awk inFile
    - cat inFile | awk -f myScript.awk

Awk script example

```
# This awk script converts data from the buildings file
# into a table seperated by tabs
BEGIN {
        FS=","
        print "Code\tName\t\tFloors"
}
/./ {
        printf "%s\t%s\t\t%d\n",$1,$2,$3
}
END {
        print "All Done!\n"
}
```

Awk variables

- Awk supports two types of variables
  - Built-In
    - Assigned to fields in a record automatically
      - Access using $0, $1 … $n
    - Provide access to options and command line arguments
      - accessed by name; OFS, FS, NF, etc...
  - User defined
    - Values are assigned using the “=” operator
      - var=56
      - var2=word
    - Unlike shell scripting, user variables are accessed by name
      - {print var2}

User defined variables

- Can contain letters, digits and underscores
- <u>must not</u> start with an integer
- case sensitive
- Can be passed into scripts from the command line
  - awk -f myScript.awk var=10 inFile
    - By default, these variables will not be available in the BEGIN block. Add the **-v** option to change this
      - awk -v var=10 -f myScript.awk inFile

User variables example (userVars.awk)

```
BEGIN {
        print "var="var
}
/./ {
        print "var="var
}
END {
        print "var="var
}
```

Built-In Variables

- Field and record delineators are controlled by Built-In variables

Name	Description	Default vlue
FIELDWIDTHS	Optionally; a space delimited list of the length of each field in number of characters. When set; IFS is ignored	
FS	Input Field Separator	whitespace characters
RS	Input Record Separator	newline
{OFS	Output Field Separator	space
ORS	Output Record Separator	newline

Built-In variables example (weekdays.awk)

weekdays.awk

```
BEGIN {
	FS=","
	RS=";"
	OFS="\t"
	ORS="\n"
	print "#","Short","Full Name"
}
$1~/[0-9]/ {
	print $1,$2,$3
}
END {}
```

weekdays.csv

```
1,Sun,Sunday;2,Mon,Monday;3,Tue,Tuesday;4,
Wed,Wednesday;5,Thu,Thursday;6,Fri,Friday;7,
Sat,Saturday
```

FIELDWIDTH example (psaux.awk)

```
# Formats output of `ps aux` into CSV
# Filters only processes running as root
BEGIN {
	FIELDWIDTHS="8 6 5 5 7 6"
	OFS=","
}
$1~/root/ {
	print $1,$2,$3,$4,$5,$6
}
END {}
```

Built-In Data Variables

Name	Description	Default vlue
NR	Contains the number of input records processed so far	
OFMT	Output format for numbers	%.6f
NF	Number of Fields in current line	
FILENAME	Name of input file	
ARGC	Number of command line arguments	
ARGV	Array of command line arguments	

Data variables example (dataVars.awk)

```
# Use with buildings file
BEGIN {
	FS=","
	OFS="\t"
}
{
	print NR,$1,$2,$3
}
END {
	print "Procesed " NR " records from file: " FILENAME
}
```

Arrays

- Awk supports associative arrays (hash maps)
- declaring arrays
  - VAR[0]=word
  - VAR[1]=another
  - VAR2=[“keyword”]=value
- Arrays from lists
  - split(LIST, DESTINATION_ARRAY, DELIMITER)
- Accessing elements
  - print VAR[1],VAR[“keyword”]

Arrays example (birthdays.awk)

```
BEGIN {
	split("Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec",MONTHS,",")
	FS=","
	OFS="\t"
}
{
	print $2,MONTHS[int($1)],$3
}
END {}
```

Expressions

- Awk supports many of the operators you are familiar with from BASH
    - +, -, *, % (modulus), /, ** (exponent)
- Also supports increment and decrement operators;
    - ++, --, +=, -+
- Example;
    - AVERAGE=($1 + $2 + $3)/3

Expression example (averageFl

```
BEGIN {
        COUNT=0
        TOTAL=0
        FS=","
        OFS="\t"
}
$3~/[0-9]/ {
        COUNT++
        TOTAL+=$3
        print $1,$2,$3
}
END {
        print "Floor Stats:"
        print "Average",TOTAL/COUNT
}
```

If statement

- Awk uses the same if, else if, else pattern we have seen in BASH; with slightly different syntax
- Conditions can use operators such as; ==, !=, <, >, <=, >=
- Example;

```
if (COUNT == 3) {
    # do something
} else if (COUNT == 0) {
    # do something else if condition was met
} else {
    # do something else
}
```

If Example (if.awk)

```
BEGIN {
        MUTI=0
        SINGLE=0
        FS=","
        OFS="\t"
       OFMT’”%.2f”
}
{
        if ($3 > 1) {
                MULTI++
        } else {
                SINGLE++
        }
                print $1,$2,$3
}
END {
        print "There are " SINGLE " single-story
and " MULTI " multi-story buildings"
}
```

For Loop

- for loop format;

```
for (variable; condition itteration) {
    # body of for loop
}
```

For loop example (numbers.awk)

```
BEGIN {
        FS=","
        OFS="\t"
        OFMT="%.2f"
        CSUM[0]=0;
        print
"#1","#2","#3","#4","#5","SUM","AVG"
}
```

```
{
        RSUM=0
        for (i=1; i <= NF; i++) {
                CSUM[int(i)] += $i
                RSUM += $i
        }
        print
$1,$2,$3,$4,$5,RSUM,RSUM/NF
}
END {
        print "--------------------------"
        print
CSUM[1],CSUM[2],CSUM[3],CSUM[4],C
SUM[5]
}
```

For loop - iterate over an array

- Similar to the for each loop in other languages
- has a modified form of the normal for loop;
    - notice the use of the keyword **in**

```
for (VAR in ARRAY) {
    print VAR
}
```

    - VAR will contain one of the associative values (or indexes) from the array ARRAY.
    - Loop will continue until there are no indexes left in the array

for each example (numbers-foreach.awk)

```
BEGIN {
	FS=","
	OFS="\t"
	OFMT="%.2f"
	CSUM[0]=0;
	print
"#1","#2","#3","#4","#5","SUM","AVG"
}
{
	RSUM=0
	for (i=1; i <= NF; i++) {
		CSUM[int(i)] += $i
		RSUM += $i
	}
	print $1,$2,$3,$4,$5,RSUM,RSUM/NF
}
```

```
END {
	print "--------------------------"
	for (NUMBER in CSUM) {
		if (NUMBER != 0) {
			print "Column " NUMBER " total
is: " CSUM[NUMBER]
		}
	}
}
```

While loop

- Like the **while** loop we used in BASH; continues to loop while the condition evaluates to true
- Format;

```
while (condition) {
    # do something while true
}
```

While loop example (numbers-while.awk)

```
BEGIN {
        FS=","
        OFS="\t"
        print
"#1","#2","#3","#4","#5","MIN","MAX"
}
```

```
{
        MIN=-1
        MAX=-1
        COL=1
        while (COL <= NF) {
                if ($COL < MIN || MIN == -1) {
                        MIN=$COL
                }
                if ($COL > MAX || MAX == -1)
{
                        MAX=$COL
                }
                COL++
        }
        print $1,$2,$3,$4,$5,MIN,MAX
}
END {}
```

User defined functions

- Awk provides several built-in functions for manipulating strings, inegers and performing mathematical operations. We can define our own functions using the **function** keyword
  - Defined outside of command blocks
  - Functions can optionally accept parameters
  - Functions can optionally return data as well
  - You can assign the output of a function to a variable

```
function addTwo (ARG1, ARG2) {
        return ARG1+ARG2
}
RESULT=addTwo(3, 4)
```

Function example (numbers-func.awk)

```
function printSep () {
        print "-----------------------------"
}
BEGIN {
        FS=","
        OFS="\t"
        OFMT="%.2f"
        CSUM[0]=0;
        print
"#1","#2","#3","#4","#5","SUM","AVG"
        printSep()
}
```

```
{
        RSUM=0
        for (i=1; i <= NF; i++) {
                CSUM[int(i)] += $i
                RSUM += $i
        }
        print
$1,$2,$3,$4,$5,RSUM,RSUM/NF
}
END {
        printSep()
        print
CSUM[1],CSUM[2],CSUM[3],CSUM[4],C
SUM[5]
}
```