Tawking AWK

PRESENTED BY:
Kent Archie
kentarchie@gmail.com

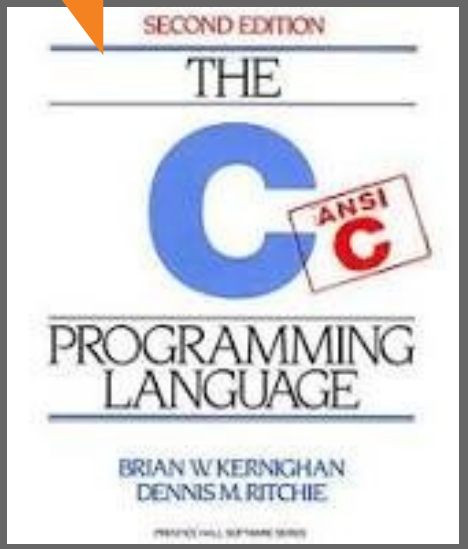
AWK

The name awk comes from the initials of its designers:

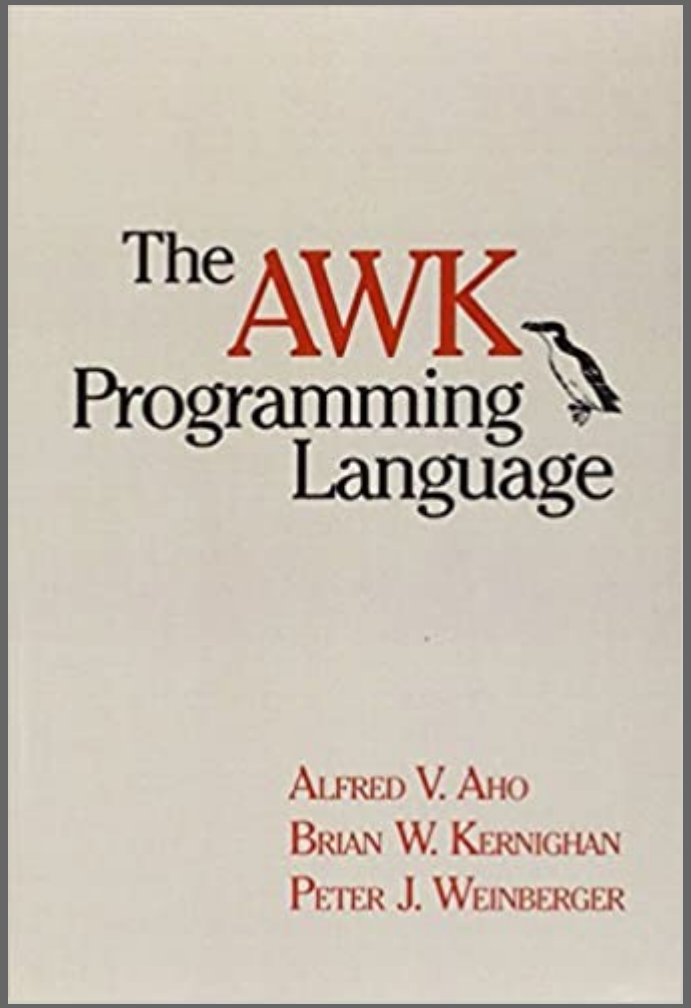
Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan.

The original version of awk was written in 1977 at AT&T Bell Laboratories.

Aho and Kernighan



Together, they wrote this book



Versions

- Linux comes with awk, nawk and usually gawk.
- Awk is the original AT&T version
- Nawk is the major rewrite from 1985
- Gawk is the GNU version, a super set of nawk
- Gawk has networking and debugging tools
- Code here uses gawk

AWK is mostly known for one liners, like

http://tuxgraphics.org/~guido/scripts/awk-one-liner.html

```
#Print decimal number as hex (prints 0x20):
gawk 'BEGIN{printf "0x%x\n", 32}'


# print section of file based on line numbers
(lines 8-12, inclusive)
gawk 'NR==8,NR==12' /etc/passwd


#Sorted list of users
gawk -F ':' '{ print $1 | "sort" }'
/etc/passwd
```

Basic structure

```
BEGIN {
  #This is run exactly once BEFORE any input
  print “before processing lines”
}
# this is run for each input line
{ print $0 } # process lines
.
END { # this is run exactly once AFTER all the input
  print “after the last line processed”
}

#This just prints the input with the two lines before and after
```

More details on structure

The BEGIN and END sections are optional.

Between them can come several other sections.

They each take the form of

```
Pattern {Action}
```

For each line read, if the pattern matches, the action is executed.

If the pattern is blank, the action is run for each line of input

The default action is to print the line

gawk 'BEGIN {print "Hello, World!";}'

gawk '{print}' shoppingData.json

gawk '$0' shoppingData.json

Default Behavior

- awk expects each line to be a separate record
- It then splits the record into fields
- Each field is assigned a variable named $1, $2 etc.
- $0 is the entire line
- The default pattern matches all lines
- The default action is to print the entire line
- FS is the input field separator, default is space
- OFS is the output field separator, default is space
- RS is the input record separator, default is newline
- ORS is the output record separator, default is newline

Example

From */etc/*passwd

```
kent:x:1000:1000:kent archie,,,:/home/kent:/bin/bash
```

We need to set the FS to ":"

Then, as each line is seen, it is already split into fields

```
$1 = kent
$2 = x
$3 = 1000
```

Example using patterns

- From earlier
- `gawk 'NR==8,NR==12'`
- No BEGIN or END
- NR is a language variable holds the current line number
- So, this is a range and matches if the line number is between 8 and 12 inclusive
- There is no code so the default action is to print the line

Using passwd file

kent:x:1000:1000:kent archie,,,:/home/kent:/bin/bash

```
gawk '
BEGIN { FS=":"; print "Name\tShell"}
/^kent/ { printf "%s\t%s\n",$5, $7}'
< /etc/passwd


gawk '
BEGIN { FS=":"; print "Name\tShell"}
!/bash/ { printf "%s\t%s\n",$1, $7}'
 < /etc/passwd
```

Get File Info

```
ls -l | gawk ‘
BEGIN { print "File\tSize\tOwner"}
{ printf “%s\t%d\t%s\n”,$9, $5, $3}
END { print " - DONE -" }’
```

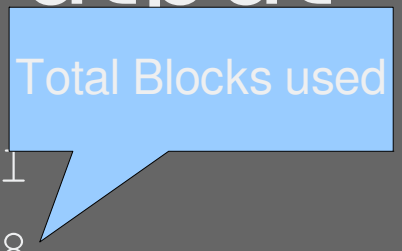
Notice there is no pattern, so all lines are printed and since the fields are separated by spaces, we don’t need to set FS

Example ls -l output
-rwxrwxr-x 1 kent kent 932 May 7 22:25 awkWeb.awk

Results

```
File Size Owner
   0
awkWeb.awk   932  kent
beta_2_a.zip 4486 kent
csv.awk 10897   kent
csvToJson.awk  1211 kent
howdy.html   108  kent
notes.txt 333  kent
sparse_csv.awk 4344 kent
tabs.vim  83 kent
 - DONE -
```

ls -l Output




```
==>ls -l
total 48
-rwxrwxr-x 1 kent kent   932 May  7 22:25 awkWeb.awk
-rw-rw-r-- 1 kent kent  4486 Apr 30 22:02 beta_2_a.zip
-rwxr-xr-x 1 kent kent 10897 Apr 30 22:55 csv.awk
-rwxrwxr-x 1 kent kent  1211 May  7 23:51 csvToJson.awk
-rw-rw-r-- 1 kent kent   108 May  7 22:28 howdy.html
-rw-rw-r-- 1 kent kent   333 Apr 30 22:28 notes.txt
-rw-rw-r-- 1 kent kent  4344 May 31  2009 sparse_csv.awk
-rw-rw-r-- 1 kent kent    83 Apr 30 22:09 tabs.vim
```

Add a pattern

Note the first line

```
total 48
```

We want to skip this

Add a pattern

The middle part

```
{ print $0 } # process lines
```

is actually

```
pattern { print $0 } # process lines
```

Add a pattern

The pattern is often a regular expression

If the line matches, the action is performed

In this case, it's simple, just look for lines that start with '-'

```
ls -l | gawk '
BEGIN { print "File\tSize\tOwner"}
/^-/ { printf "%s\t%d\t%s\n",$9, $5, $3}
END { print " - DONE -" }'
```

Results

```
File Size Owner
awkWeb.awk  932  kent
beta_2_a.zip  4486 kent
csv.awk  10897  kent
csvToJson.awk 1211 kent
howdy.html  108  kent
notes.txt 333  kent
sparse_csv.awk  4344 kent
tabs.vim  83 kent
 - DONE -
```

Question

What happens if there are links?

```
total 96
lrwxrwxrwx 1 kent kent    24 Aug 17 17:23 1939 ->
../data/WeatherData/1939
-rwxr-xr-x 1 kent kent  8616 Aug 16 23:17 2darray
-rwxrwxr-x 1 kent kent   771 Aug 16 23:18 2darray1.awk
-rw-rw-r-- 1 kent kent   824 Aug 16 23:17 2darray.c
-rw-r--r-- 1 kent kent   479 Aug 15 15:52 apache.awk
-rwxrwxr-x 1 kent kent   932 May  7 22:25 awkWeb.awk
-rwxr-xr-x 1 kent kent 10897 Apr 30 22:55 csv.awk
-rwxrwxr-x 1 kent kent  1720 May 21 23:06
csvToJson.awk
-rw-rw-r-- 1 kent kent   562 Aug 17 17:44 examples.txt
-rw-rw-r-- 1 kent kent   108 May  7 22:28 howdy.html
-rwxr-xr-x 1 kent kent   206 May  9 20:10 lsfilter.awk
-rwxr-xr-x 1 kent kent   317 May  9 22:13 lsfilter.sh
```

Results2

```
> BEGIN { print "File\tSize\tOwner"}
> /^-/ { printf "%s\t%d\t%s\n",$9, $5, $3}
> END { print " - DONE -" }'

File  Size  Owner
2darray 8616  kent
2darray1.awk  771   kent
2darray.c  824   kent
apache.awk 479   kent
awkWeb.awk 932   kent
csv.awk 10897 kent
csvToJson.awk 1720  kent
examples.txt  679   kent
howdy.html 108   kent
lsfilter.awk  206   kent
lsfilter.sh   317   kent
notes.txt  333   kent
samplePlot.txt   107   kent
sparse_csv.awk   4344  kent
```

```
lrwxrwxrwx 1 kent kent    24 Aug 17 17:23 1939 -> ../
data/WeatherData/1939
Is missing
```

Two Solutions

```
# check for lines starting with either - or l
ls -l | gawk '
BEGIN { print "File\tSize\tOwner"}
/^-/ || /^l/ { printf "%s\t%d\t%s\n",$9, $5, $3}
END { print " - DONE -" }'

#check for lines that don’t start with total
ls -l | gawk '
BEGIN { print "File\tSize\tOwner"}
!/^.*total/ { printf "%s\t%d\t%s\n",$9, $5, $3}
END { print " - DONE -" }'
```

Bash Version

```
echo -e "File\tSize\tOwner"
ls -l | egrep -s '^-' | tr -s " " | cut -d' ' -f9,5,3
echo " - DONE -"
```

```
File  Size  Owner
kent 932 awkWeb.awk
kent 4486 beta_2_a.zip
kent 10897 csv.awk
kent 1211 csvToJson.awk
kent 108 howdy.html
kent 83 lsfilter.sh
kent 333 notes.txt
kent 4344 sparse_csv.awk
kent 83 tabs.vim
 - DONE -
```

Note the column order is wrong

Bash version 2

```
echo -e "File\tSize\tOwner"

 ls -l | egrep -s '^-' | tr -s " " |  while read -r c1 c2 c3 c4 c5 c6
c7 c8 c9
    do
       echo $c9 $c5 $c3
   done
echo " - DONE -"
```

```
File  Size  Owner
awkWeb.awk 932 kent
beta_2_a.zip 4486 kent
csv.awk 10897 kent
csvToJson.awk 1211 kent
howdy.html 108 kent
lsfilter.sh 203 kent
notes.txt 333 kent
sparse_csv.awk 4344 kent
tabs.vim 83 kent
 - DONE -
```

Added up the sizes (AWK)

```
 1  ls -l | gawk '
 2  BEGIN {
 3     print "File\tSize\tOwner";
 4     totalSize = 0;
 5  }
 6
 7   /^-/ {
 8     printf "%s\t%d\t%s\n",$9, $5, $3;
 9     totalSize += $5;
10  }
11
12  END {
13     printf "total size = %d\n",totalSize;
14     print " - DONE -"
15  }
```

sumSizes.awk

Added up the sizes (Bash)

```
 1 : #!/bin/bash
 2 : echo -e "File\tSize\tOwner"
 3 : totalSize=0
 4 : ls -l | egrep -s '^-' | tr -s " " |
 5 : {
 6 : while read -r c1 c2 c3 c4 c5 c6 c7 c8 c9
 7 : do
 8 :   echo $c9 $c5 $c3
 9 :   totalSize=`echo "$c5 + $totalSize" | bc`
10 : done
11 :  echo "total size = $totalSize"
12 :  echo " - DONE -"
13 : }
```

sumSizes.sh

There are surely better ways to do some of this

Just a cool thing you can do

```
 1 #!/usr/bin/gawk -f
 2 BEGIN {
 3 if (ARGC < 2) { print "Usage: awkWeb  file.html"; exit 0 }
 4    Concnt = 1;
 5         while (1) {
 6         RS = ORS = "\r\n";
 7         HttpService = "/inet/tcp/8080/0/0";
 8         getline Dat < ARGV[1];
 9         Datlen = length(Dat) + length(ORS);
10         while (HttpService |& getline ){
11        if (ERRNO) { print "Connection error: " ERRNO; exit 1}
12                 print "client: " $0;
13                 if ( length($0) < 1 ) break;
14          }
15         print "HTTP/1.1 200 OK"              |& HttpService;
16         print "Content-Type: text/html"      |& HttpService;
17         print "Server: wwwawk/1.0"           |& HttpService;
18         print "Connection: close"            |& HttpService;
19         print "Content-Length: " Datlen ORS |& HttpService;
20         print Dat                            |& HttpService;
21         close(HttpService);
22         print "OK: served file " ARGV[1] ", count " Concnt;
23         Concnt++;
24       }
25 }
```

awkWeb.awk

Associative arrays

- AWK supports arrays with non-numeric indices
- arr[0] = 7 works sort of like other languages
- But you can also do
- arr[“zero”] = 7
- No separate initialization
- Similar to C#, Java maps, Python dict
- Standard awk has no direct multi-dimensional arrays
- arr[1 "," 2] = 7;
- Same as doing arr[“1,2”]=7

Associative Arrays (2)

- Arr[5]=7 creates an array with exactly one element
- So length(arr) == 1, not 5
- You loop over an array like this
- for (i in arr) print arr[i]
- They aren't really arrays, since not a fixed length or type.
- Because they are like a hash table, elements might not appear in the order you entered them

Associative Arrays Example

```
1  gawk '
2  BEGIN {
3     arr[5] = 5;
4     arr[1] = 1;
5     arr[6] = 6;
6     arr["six"] = "six";
7     for(i in arr) print i, arr[i];
8  }
9  '
```

```
six six
1 1
5 5
6 6
```

LoopTest.awk

Gawk Arrays Of Arrays

A few quick examples

```
A[1][1] = 1
A[1][2] = 2
B[1][3][1, "name"] = "barney"
Think of A like this
A[1] = [1,2]
```

Read a file of plot data

2dArrayExample.awk

```
# input data format 4.2,215
   1  BEGIN {
   2     FS=","
   3     row = 1;
   4  }

   5  {
   6     data[row][1] = $1;
   7     data[row++][2] = $2;
   8  }
   9
  10  END {
  11   print("\nlooping print\n");
  12   for( i in data)
  13     for( j in data[i])
  14        printf("data[%d][%d] = :%5.1f\n",
                 i,j,data[i][j]);
  15  }
```

Results

looping print

data[1][1] = : 4.2:
data[1][2] = :215.0:
data[2][1] = : 16.4:
data[2][2] = :325.0:
data[3][1] = : 11.9:
data[3][2] = :185.0:
data[4][1] = : 15.2:
data[4][2] = :332.0:
data[5][1] = : 18.5:
data[5][2] = :406.0:
data[6][1] = : 22.1:
data[6][2] = :522.0:
data[7][1] = : 19.4:
data[7][2] = :412.0:
data[8][1] = : 25.1:
data[8][2] = :614.0:
data[9][1] = : 23.4:
data[9][2] = :544.0:
data[10][1] = : 18.1:
data[10][2] = :421.0:
data[11][1] = : 22.6:
data[11][2] = :445.0:
data[12][1] = : 17.2:
data[12][2] = :408.0:

C Version

2darray.c

```
# input data format 4.2,215

 1 float disp[12][2];
 2 int row=0;
 3 char *pt;
 4 while (fgets(str, MAXCHAR, fp) != NULL) {
 5    pt = strtok (str,",\n");  // split on comma
 6   disp[row][0] =  atof(pt);
 7   pt = strtok (NULL, ",\n");
 8   disp[row++][1] =  atof(pt);
 9 }  // reading lines
10 fclose(fp);
11 //Displaying array elements
12 printf("\nTwo Dimensional array elements:\n");
13 for(int i=0; i<12; i++) {
14   for(int j=0;j<2;j++) {
15      printf("disp[%d][%d] = :%5.1f:\n",
            i,j,disp[i][j]);
16   }
17 }
```

Functions

- User define functions are allowed
- Similar syntax to C

```
function foo(a,b) {
   return a+b;
}
```

Get Files Types

Types.awk

```
 1 : ls -l | gawk '
 2 : # get the extension part of a file name
 3 : function getExtension(file)
 4 : {
 5 :    n = split(file, a, "."); # split the file name
into parts
 6 :    return(a[n]);  # last element of the array
 7 : } # getExtension
 8 :
 9 : BEGIN {
10 :   print "File\tType";
11 : }
12 :
13 : /^-/ {
14 :   type = getExtension($9);
15 :   types[type] += 1;
16 : }
17 :
18 : END {
19 :   for (t in types) {
20 :      printf("%s\t%d\n", t,types[t]);
21 :   }
22 : }'
```

File Types Results

```
==>./types.awk
File Type
html 1
zip  1
awk  6
txt  1
vim  1
sh 1
```

Run commands and get results

typesGetline.awk

```
1   gawk '
2   # get the extension part of a file name
3   function getExtension(file)
4   {
5      n = split(file, a, "."); # split the file name into parts
6      return(a[n]);  # last element of the array
7   } # getExtension
8
9   BEGIN {
10        print "File\tType";
11        # get list of file, skip the total line and squeeze the spaces
12        cmd = "ls -l | egrep '^-'| tr -s \" \" ";
13     while ( ( cmd | getline result ) > 0 )
14        {
15        n = split(result, parts, " ");
16              # lines look like
17              # -rwxrwxr-x 1 kent kent 932 May 7 22:25 awkWeb.awk
18              # file name is the last field
19              type = getExtension(parts[length(parts)]);
20              types[type] += 1;
21     }
22     close(cmd);
```

Run commands and get results con’t

```
23          for (t in types) {
24                printf("%s\t%d\n", t,types[t]);
25          }
26          print " - DONE -"
27    }
28    '
```

Some one-liners

A collection of one-liners
https://www.pement.org/awk/awk1line.txt

Explanation of the one-liners
https://catonmat.net/awk-one-liners-explained-part-one

```
awk '1; { print "" }' # print file double spaced
```

```
# custom line numbers
awk '{ printf("%5d : %s\n", NR, $0) }' filename
    1 : set term png truecolor
```

```
#count lines containing pattern
awk '/if/ { n++ }; END { print n+0 }' shopPlot.awk
```

Some More One-Liners

```
#trim whitespace
gawk '{ gsub(/^[ \t]+|[ \t]+$/, ""); print }' filename


#replace "foo" with "bar" on lines that contain "baz".
gawk '/baz/ { gsub(/foo/, "bar") }; { print }'


#Remove duplicate, nonconsecutive lines.
gawk '!a[$0]++'

#print the line before the matching line
gawk '/regex/ { print x }; { x=$0 }'

# handle the first line matching
gawk '/regex/ { print (x=="" ? "match on line 1" :
x) }; { x=$0 }'
```

Gather Spending Totals

shopPlot/shopPlot.awk

```
 1  #!/usr/bin/gawk -f
 2  @include "../lib/csv.awk" # from http://lorance.freeshell.org/csv/
 3  @include "../utilities.awk"
 4
 5  BEGIN { #run once before processing lines
 6     FS=",";
 7  } # BEGIN
 8
 9  FNR == 1 {next} # skip first line
 10
 11 FNR != 1{
 12    if(NR % 100 == 0) printf("Lines so far (%d)\n", NR);
 13
 14    num_fields = csv_parse($0, csv, ",", "\"", "\"", "\\n", 0)
 15    if (num_fields < 0) {
 16       printf("ERROR: %d (%s) -> %s\n", num_fields,
csv_err(num_fields), $0);
 17       continue;
 18    }
 19    totals[csv[1]] += csv[4];
 20
 21 } # for each line
 22
 23 END { # run once after processing lines
 24       walk_array(totals, "totals", I);
 25       printf("END: processed %d data points\n",NR);
 26 } # END
```

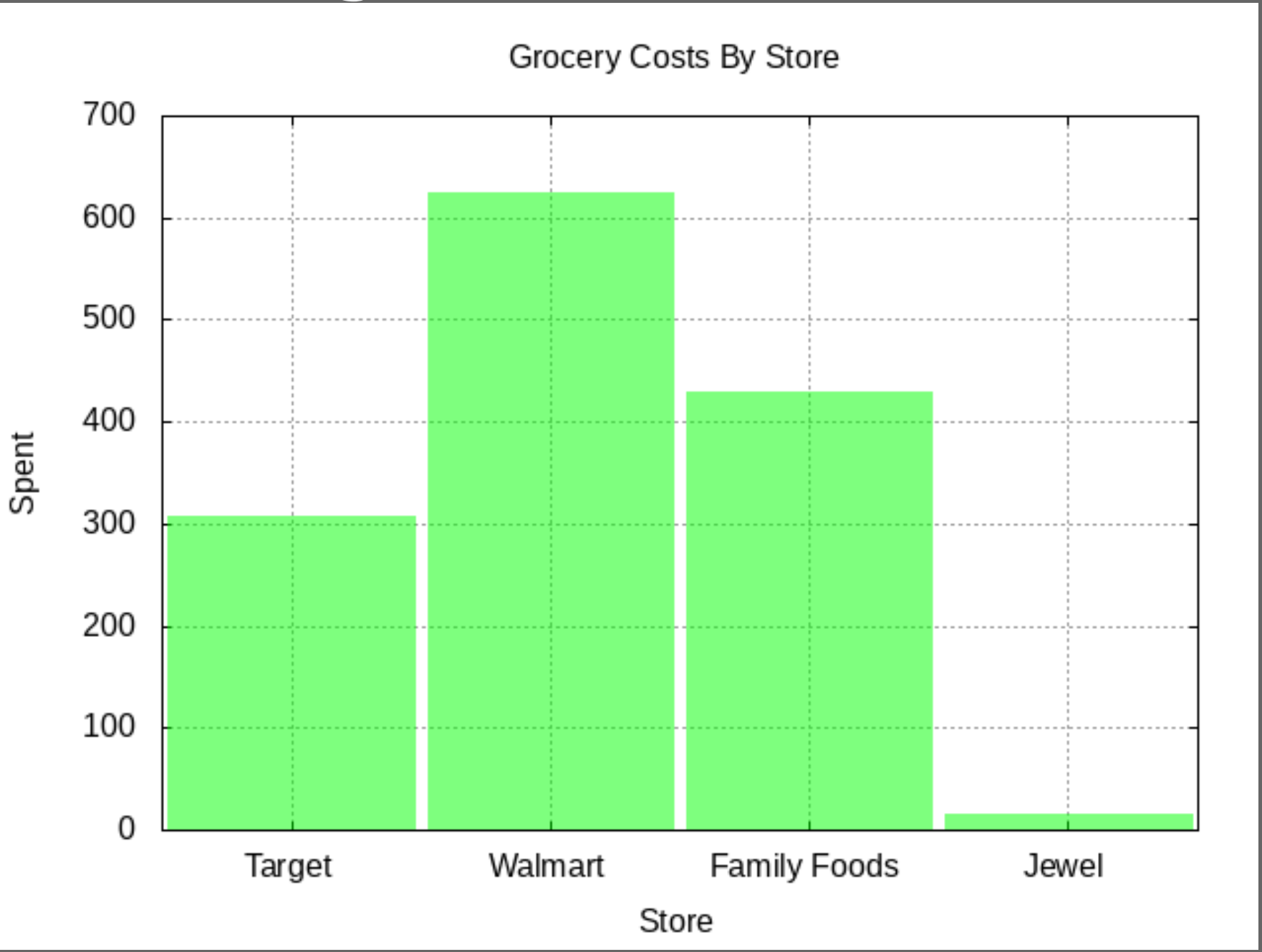
Code to plot the totals

```
23  END { # run once after processing lines
 24       walk_array(totals, "totals", I);
 25       printf("END: processed %d data points\n",NR);
 26
 27       system("rm -f shopPlot.dat");
 28
 29       printf("Store\tTotal\n") > "shopPlot.dat"
 30
 31       for(t in totals) {
 32          printf("\"%s\" %f\n",t,totals[t]) >>
"shopPlot.dat"
 33       }
 34
 35       system("gnuplot -c testPlot.txt 2>&1");
 36     } # END
```

TestPlot.txt

```
set term png truecolor
set output "testPlot.png"
set xlabel "Store"
set ylabel "Spent"
set title "Grocery Costs By Store"
set grid
set boxwidth 0.95 relative
set style fill transparent solid 0.5 noborder
plot "shopPlot.dat" using 2:xticlabels(1) with boxes
lc rgb"green" notitle
```

Resulting Plot



AWK References

https://www.gnu.org/software/gawk/manual/gawk.pdf
GAWK: Effective AWK Programming

https://www.grymoire.com/Unix/Awk.html
Pretty good tutorial

https://www.ncei.noaa.gov/data/global-summary-of-the-day/
Sample Weather Data

https://www.pement.org/awk/awk1line.txt
AWK one liners

https://catonmat.net/awk-one-liners-explained-part-one
https://catonmat.net/awk-one-liners-explained-part-two
https://catonmat.net/awk-one-liners-explained-part-three
Detailed explanations of the one liners

Other ways to manipulate data

- q
- Allows you to run SQL queries against CSV files.
- https://harelba.github.io/q/

```
q -H  -d","  "SELECT item,store,price FROM
shoppingData.csv where item like '%milk%'"
Milk,Family Foods,2.59
milk,Family Foods,5.18
Milk,Family Foods,2.59
Milk,Family Foods,2.59
Milk,Target,3.19
Milk,Walmart,4.58
Milk,Walmart,4.58
Milk,Jewel,2.59
Milk,Family Foods,2.59
Milk,Family Foods,3.79
Milk,Family Foods,3.79
Milk,Family Foods,3.79
```

```
awk -F"," '$3 ~ /.*Milk.*/ { print $3,$1,$4}'
shoppingDataExample/shoppingData.csv
```

JSON filter language

https://stedolan.github.io/jq/tutorial/

It is a way to extract, and combine JSON records
The syntax is a little confusing but the tutorial above has many examples

To just sort of prettyprint the records, do

jq "." < shoppingData.json

This prints all the records


```
  {
    "store": "Family Foods",
    "date": "2014-06-14",
    "item": "Salsa",
    "price": 2.79,
    "categories": "Condiments"
  },
```

jq examples

Get a certain value from each record

jq ".[].price" shoppingData.json

...
2.99
0.43
2.88
3.99
3.99
3.79
0.21
3.79
...

jq examples

Extract a subset of fields

jq '.[] | {store:.store,item:.item}' shoppingData.json

```
{
  "store": "Family Foods",
  "item": "Garlic"
}
{
  "store": "Family Foods",
  "item": "Tax"
}
{
  "store": "Family Foods",
  "item": "Savings"
}
```

Notice there are no commas between records, so not a JSON array

jq examples

This gets the results as an array

```
jq '[.[] |  {store:.store,item:.item}]' shoppingData.json
```

```
[
...
{
    "store": "Family Foods",
    "item": "Bread Crumbs"
  },
  {
    "store": "Family Foods",
    "item": "Garlic"
  },
  {
    "store": "Family Foods",
    "item": "Tax"
  },
...
]
```

Questions?

CONTACT:
kentarchie@gmail.com