Instructor: Nicolas Savva

February 13, 2015

1 based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years
Announcements

- A3 (due 02/20)
- February break (No Monday lecture 02/16)
- OH resume on Wednesday
AWK is a programming language designed for processing text-based data
- allows us to easily operate on fields rather than full lines
- works in a *pattern-action* matter, like *sed*
- supports numerical types (and operations) and control flow (*if-else* statements)
- extensively uses string types and associative arrays

Created at Bell Labs in the 1970s
- by Alfred Aho, Peter Weinberger, and Brian Kernighan

An ancestor of *Perl*
- and a cousin of *sed* :-P

Very powerful
- actually *Turing Complete*
gawk

- gawk is the GNU implementation of the AWK programming language. On BSD/OS X the command is called awk.
- AWK allows us to setup filters to handle text as easily as numbers (and much more)

The basic structure of a awk program is

```plaintext
pattern1  {  commands  }
pattern2  {  commands  }
...
```

- patterns can be regular expressions! Gawk goes line by line, checking each pattern one by one and if it’s found, it performs the command.
Why gawk and not sed

- convenient numerical processing
- variables and control flow in the actions
- convenient way of accessing fields within lines
- flexible printing
- built-in arithmetic and string functions
gawk '/[Mm]onster/ {print}' Frankenstein.txt

gawk '/[Mm]onster/' Frankenstein.txt

gawk '/[Mm]onster/ {print $0}' Frankenstein.txt

- All print lines of Frankenstein containing the word Monster or monster.
- If you do not specify an action, gawk will default to printing the line.
- $0 refers to the whole line.
- gawk understands extended regular expressions, so we do not need to escape +, ?, etc.
Gawk allows blocks of code to be executed only once, at the beginning or the end.

gawk 'BEGIN {
   print "Starting search for a monster"}
   /[Mm]onster/ { count++}
END {
   print "Found " count " monsters in the book!"
   }
' Frankenstein.txt

gawk does not require variables to be initialized

integer variables automatically initialized to 0, strings to "".
The real power of gawk is its ability to automatically separate each input line into fields, each referred to by a number.

```
gawk
BEGIN {print "Beginning operation"; myval = 0}
/debt/ { myval -= $1}
/asset/ { myval += $1}
END { print myval}’ infile
```

- $0 refers to the whole line
- $1, $2, ... $9, $(10) ... refer to each field
- The default Field Separator (FS) is white space.
If no pattern is given, the code is executed for every line:

```
gawk ' {print $3 }' infile
```

Prints the third field/word on every line.
Other gawk variables

- **NF** - # of fields in the current line
- **NR** - # of lines read so far
- **FILENAME** - the name of the input file

```gawk
{for (i=1;i<=NF;i++) print $i }
```

in `infile`

Prints all words in a file

- You **cannot** change NF or NR.
Let's implement `wc -l` in awk!
The field separator

- FS - The field separator
- Default is " "

```
gawk 'BEGIN { FS = "",""} {print $2 }' infile
```
- gawk -F: also allows us to set the field separator
gawk can match any of the following pattern types:

- /regular expression/
- relational expression
- exp && exp
- exp || exp
- condition ? statement1 : statement2 - if condition, then statement1, else statement2
- ! exp
- and more...
gawk '($1 > .5){print $2 }' infile

Other relational operators

- <, <=, >, >=, !=, ==
gawk can match any of the following pattern types:

- `/regular expression/`
- relational expression
- `pattern && pattern`
- `pattern || pattern`
- `pattern1 ? pattern2 : pattern3` - if `pattern1`, then match `pattern2`, if not then match `pattern3`
- `(pattern)` - to change order of operations
- `! pattern`
- `pattern1, pattern2` - match `pattern1`, work on every line until it matches `pattern2` (cannot combine this one)
The field separator revisited

- **FS** - The field separator
- Default is "  

```gawk
BEGIN { FS = ":" }
toupper($1) ~ /FOO/ { print $2 } ' infile
```

- `gawk -F:` also allows us to set the field separator
- `toupper()`, `tolower()` - built in functions
- `~` - gawk matching command
- `!~` - gawk not matching command
Other gawk functions

- exp(x) : exponential of x
- rand() : produces a random number between 0 and 1
- length(x) : returns the length of x
- log(x) : returns the log of x
- sin(x) : returns the sin of x
- int(x) : returns the integer part of x
What type of code can I use in gawk?

Gawk coding is very similar to programming in C.

- `for(i = ini; i <= end; increment i) {code}`
- `if (condition) {code}
  (In both cases the {} can be removed if only one command is executed)
- and so on. See the gawk manual for more

Variables and Associative Arrays

- gawk handles variable conversion automatically

\[ \text{total} = 2 + "3" \] assigns 5

- Arrays are automatically created and resized
- Arrays are "associative", meaning the index can be any string:

\[
\text{array["txt"]} = \text{value}
\]

array[50] is equivalent to array["50"].
Variables

- gawk handles variable conversion automatically

```plaintext
total = 2 + "3"  // assigns 5
total++  // total = total + 1
++total  // returns current value, then total = total + 1
line = "foo" "bar"  // concatenates two strings
line = var "bar"  // concatenates the contents of var with bar
```
Operators

- ++ Add 1 to variable.
- -- Subtract 1 from variable.
- += Assign result of addition.
- -= Assign result of subtraction.
- *= Assign result of multiplication.
- /= Assign result of division.
- %= Assign result of modulo.
- **= Assign result of exponentiation
Another gawk function

- `substr(string, beg[, len])`: Return substring of string at beginning position `beg` (counting from 1), and the characters that follow to maximum specified length `len`. If no length is given, use the rest of the string.
array[key1, key2, ...]
This is not what AWK is doing

<table>
<thead>
<tr>
<th>a</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ a[3][7] \]
This is not what AWK is doing either
array[3, 6]

- Multidimensional subscripts are individual strings concatenated.
- "3" and "6" in the example are concatenated together separated by the value of the system variable SUBSEP
(key, value) addition

- Arrays are automatically created and resized
- "associative" means that the index can be any string:

```javascript
array["txt"] = value
array[50] is equivalent to array["50"].
```
(key, value) modification

array["txt"]++
array["txt"]+= $1
array["txt"]+= $1 "bar"
(key, value) lookup

print array["txt"]
array["txt"] = array["txt"] "bar"
(key, value) deletion

delete array["txt"]
The following are very helpful:

```bash
if (someValue in theArray) {
    action to take if somevalue is in theArray
} else {
    an alternate action if it is not present
}

for (i in theArray) print i
```
Inverts all strings in the file
Suppose we have an iou file of the following form:

Who owes me what as of today
Name \tab Amount
Name \tab Amount

Lets write a gawk script to add up how much everyone owes us
Associative Array Example

```
gawk '  
    BEGIN {FS = "\t"}
    NR > 1 { Names[$1]+=$2 }
    END { for(i in Names) print i " owes me " Names[i] " Dollars."}
  ' ioufile

(Can you spot the error?)
```
printf("Hello World\n")

printf("%d\t%s\n", $5, $9)

where

- %d: decimal integer
- %s: string
- \t: tab
- \n: new line
\[ n = \text{split(string, array, separator)} \]

- Splits fields of string separated by separator and places them into array.
- \( n \) is the resulting number of fields
- default separator is whitespace
if ((i, j) in array)

  This tests whether the key i SUBSEP j exists in the array.
That makes life a little harder!

for (item in array)

- Each item has the form $i \text{ SUBSEP } j$
- You must use `split()` to extract individual subscript components.

```plaintext
n = split(item, subscr, SUBSEP)
subscr[1] # first component
subscr[2] # second component
... 
subscr[n] # n-th component
```
Length of an Array

- `awk 'BEGIN {A= "Ithaca is Gorges"; print length(A)}'`
  
  prints "16"

- `awk 'BEGIN {split("Ithaca is Gorges",A); print length(A)}'`
  
  prints "3"
We have only touched on the very basic things you can do with gawk to give you a taste.

Check the website for much more:

www.gnu.org/software/gawk/manual
Reminder

No lecture on Monday
Have a good February break!