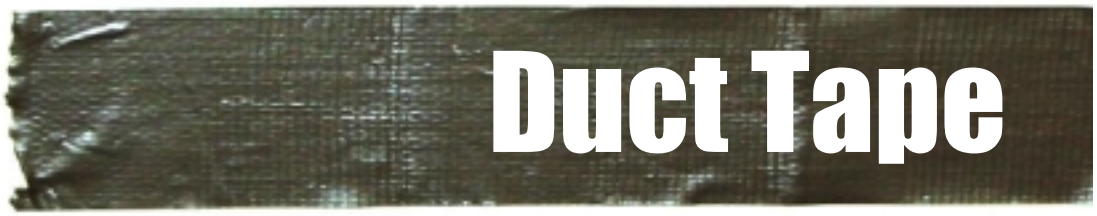
AWK: The Duct Tape of Computer Science Research

Tim Sherwood
UC Santa Barbara



- Systems Research Environment
  - Lots of simulators, data, and analysis tools
  - Since it is research, nothing works together
- Unix pipes are the ducts
- Awk is the duct tape
  - It's not the "best" way to connect everything
  - Maintaining anything complicated problematic
  - It is a good way of getting it to work quickly
    - In research, most stuff doesn't work anyways
  - Really good at a some common problems

- My Goals for this tutorial
  - Basic introduction to the Awk language
  - Discuss how it has been useful to me
  - Discuss some the limits / pitfalls
- What this talk is not
  - A promotion of all-awk all-the-time (tools)
  - A perl vs. awk battle

- Background and History
- When “this is a job for AWK”
- Programming in AWK
  - A running example
- Other tools that play nice
- Introduction to some of my AWK scripts
- Summary and Pointers

- Developed by
  - Aho, Weinberger, and Kernighan
  - Further extended by Bell
  - Further extended in Gawk
- Developed to handle simple data-reformatting jobs easily with just a few lines of code.
- C-like syntax
  - The K in Awk is the K in K&R
  - Easy learning curve

- Smart grep
  - All the functionality of grep with added logical and numerical abilities
- File conversion
  - Quickly write format converters for text files
- Spreadsheet
  - Easy use of columns and rows
- Graphing/tables/tex
- Gluing pipes

Running gawk

- Two easy ways to run gawk
- From the Command line
  - `cat file | gawk '(pattern){action}'`
  - `cat file | gawk -f program.awk`
- From a script (recommended)

```
#!/usr/bin/gawk –f
# This is a comment
(pattern) {action}
…
```

- Programming is done by building a list of rules
- The rules are applied sequentially to each record in the input file or stream
  - By default each line in the input is a record
- The rules have two parts, a pattern and an action
- If the input record matches the pattern, then the action is applied

(pattern1) { *action* }
(pattern2) { *action* }
...

Input	PING dt033n32.san.rr.com (24.30.138.50): 56 data bytes 64 bytes from 24.30.138.50: icmp_seq=0 ttl=48 time=49 ms 64 bytes from 24.30.138.50: icmp_seq=1 ttl=48 time=94 ms 64 bytes from 24.30.138.50: icmp_seq=2 ttl=48 time=50 ms 64 bytes from 24.30.138.50: icmp_seq=3 ttl=48 time=41 ms … ----dt033n32.san.rr.com PING Statistics---- 1281 packets transmitted, 1270 packets received, 0% packet loss round-trip (ms) min/avg/max = 37/73/495 ms
Program	(/icmp_seq/) {print $0}
Output	64 bytes from 24.30.138.50: icmp_seq=0 ttl=48 time=49 ms 64 bytes from 24.30.138.50: icmp_seq=1 ttl=48 time=94 ms 64 bytes from 24.30.138.50: icmp_seq=2 ttl=48 time=50 ms 64 bytes from 24.30.138.50: icmp_seq=3 ttl=48 time=41 ms

- Awk divides the file into records and fields
  - Each line is a record (by default)
  - Fields are delimited by a special character
    - Whitespace by default
    - Can be change with
      "–F" (command line) or
      FS (special varaible)
- Fields are accessed with the '$'
  - $1 is the first field, $2 is the second…
  - $0 is a special field which is the entire line
  - NF is a special variable that is equal to the number of fields in the current record

Input	PING dt033n32.san.rr.com (24.30.138.50): 56 data bytes 64 bytes from 24.30.138.50: icmp_seq=0 ttl=48 time=49 ms 64 bytes from 24.30.138.50: icmp_seq=1 ttl=48 time=94 ms 64 bytes from 24.30.138.50: icmp_seq=2 ttl=48 time=50 ms 64 bytes from 24.30.138.50: icmp_seq=3 ttl=48 time=41 ms … ----dt033n32.san.rr.com PING Statistics---- 1281 packets transmitted, 1270 packets received, 0% packet loss round-trip (ms) min/avg/max = 37/73/495 ms
Program	(/icmp_seq/) {print $7}
Output	time=49 time=94 time=50 time=41

- Variables uses are naked
  - No need for declaration
  - Implicitly set to 0 AND Empty String
- There is only one type in awk
  - Combination of a floating-point and string
  - The variable is converted as needed
    - Based on it's use
  - No matter what is in x you can always
    - x = x + 1
    - length(x)

Input	PING dt033n32.san.rr.com (24.30.138.50): 56 data bytes 64 bytes from 24.30.138.50: icmp_seq=0 ttl=48 time=49 ms 64 bytes from 24.30.138.50: icmp_seq=1 ttl=48 time=94 ms 64 bytes from 24.30.138.50: icmp_seq=2 ttl=48 time=50 ms 64 bytes from 24.30.138.50: icmp_seq=3 ttl=48 time=41 ms …

Program	(/icmp_seq/) {
	n = substr($7,6);
	printf("%s\n", n/10); #conversion
	}

Output	4.9 9.4 5.0 4.1 …

- Some built in variables
  - Informative
    - NF = Number of Fields
    - NR = Current Record Number
  - Configuration
    - FS = Field separator
- Can set them externally
  - From command line use

    Gawk –v var=value

- Patterns can be
  - Empty: match everything
    - **`{print $0}`** will print every line
  - Regular expression: **`(/regular expression/)`**
  - Boolean Expression: **`($2=="foo" && $7=="bar")`**
  - Range: **`($2=="on" , $3=="off")`**
  - Special: BEGIN and END

- All arrays in awk are associative
  - A[1] = “foo”;
  - B[“awk talk”] = “pizza”;
- To check if there is an element in the array
  - Use “in”: **`If ( “awk talk” in B )`** …
- Arrays can be sparse, they automatically resize, auto-initialize, and are fast (unless they get huge)
- Built in array iterator “in”
  - For (x in myarray) {
  - Not in **any** order

Associative Arrays

- The arrays in awk can be used to implement almost any data structure
  - Set:
    - myset[“a”]=1; myset[“b”]=1;
    - If (“b” in myset)
  - Multi-dimensional array:
    - myarray[1,3] = 2; myarray[1,”happy”] = 3;
  - List:
    - mylist[1,”data”]=2; mylist[1,”next”] = 3;

Input	PING dt033n32.san.rr.com (24.30.138.50): 56 data bytes 64 bytes from 24.30.138.50: icmp_seq=0 ttl=48 time=49 ms …

Program	(/icmp_seq/) {
	n = int(substr($7,6)/10);
	hist[n]++; #array
	}
	END {
	for(x in hist)
	printf(“%s: %s”, x*10, hist[x]);
	}

Output	40: 441 50: 216 … 490: 1

Built-in Functions

- Numeric:
  - cos, exp, int, log, rand, sqrt …
- String Functions
  - Gsub(regex, replacement, target)
  - Index(searchstring, target)
  - Length(string)
  - Split(string, array, regex)
  - Substr(string, start, length=inf)
  - Tolower(string)

Writing Functions

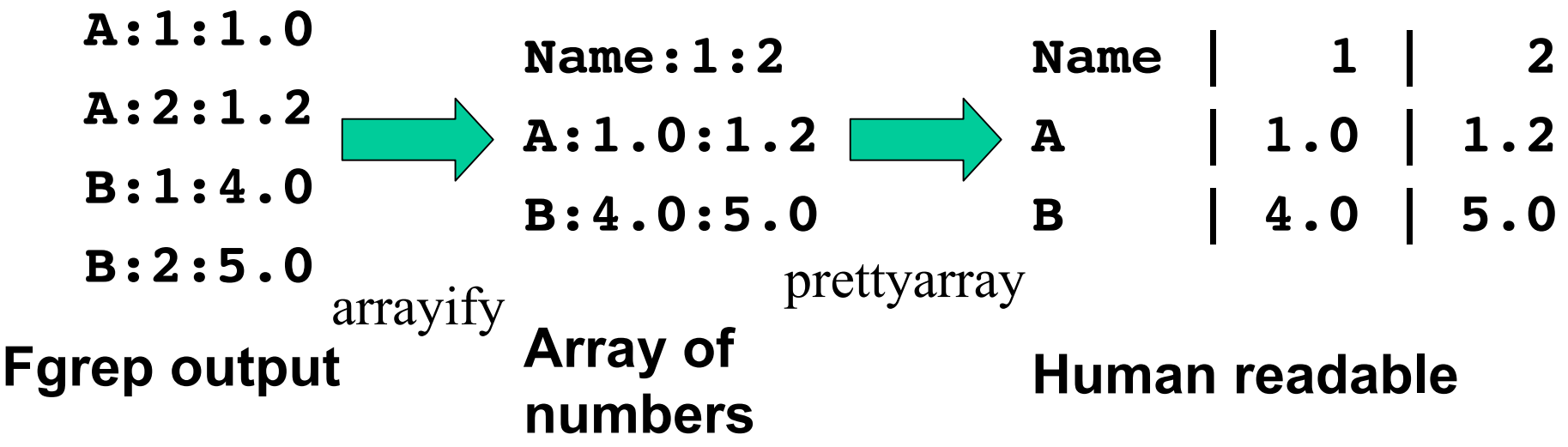
- Functions were not part of the original spec
  - Added in later, and it shows
  - Rule variables are global
  - Function variables are local

```
function MyFunc(a,b,   c,d) {
   Return a+b+c+d
}
```

- Awk is best used with pipes
- Other tools that work well with pipes
  - Fgrep: fgrep mystat *.data (parse with –F:)
  - Uniq: uniq –c my.data
  - Sort
  - Sed/tr: (handy for search and replace)
  - Cut/paste: (manipulating columns in data)
  - Jgraph/Ploticus

My Scripts

- Set of scripts for handling data files



- From the array files, my scripts will generate simple HTML tables or TeX tables, transpose the array, and other things.

Some Pitfalls

- White space
  - No whitespace between function and ‘(‘
    - Myfunc($1) = ☺
    - Myfunc ($1) = ☹
  - No line break between pattern and action
- **Don’t forget the -f on executable scripts**
  - This will just die silently… very common mistake
- No built in support for hex
  - On my web page there are scripts for that too

- Awk is a very powerful tool
  - If properly applied
  - It is not for everything (I know)
- Very handy for pre-processing
- Data conversion
- It's incrementally useful
  - Each step of the learning curve is applicable that day.
- Thank you