

Part Four

Columnar data: AWK with a
side of sort and join

What is AWK?

- AWK is a full programming language
 - variables and perl-like hashes
 - loops and conditional statements
- Has sed-like addressing and regex
- Automatically splits lines into words
- Optimized for text parsing

Terms

- ❖ **record** - like a line of input in sed, but may have a separator other than newline
- ❖ **field** - records are split into fields
- ❖ **command** - a **condition/procedure** pair
- ❖ **condition** - a logical test
- ❖ **procedure** - code block that is run if the condition is TRUE

AWK Pseudocode

```
BEGIN { do initial stuff }  
for each record in input  
    split record into fields  
    for each command  
        if condition is TRUE  
            do procedure  
END { do final stuff }
```

Outline

1. Condition statements
2. Procedure statements
3. Language structure

1. Condition Statements

Condition only calls

AWK Rule 1: If the *command* consists only of a *condition*, the *procedure* defaults to print *record*.

Unlike sed, AWK does not print by default

Simple Examples

Print lines matching 'Here' (like grep)

\$ awk '/Here/'

Print from 'Here' to 'There'

\$ awk '/Here/,/There/'

AWK knows extended regular expressions

\$ awk '/^AT[0-9]+G[0-9]+/'

AWK Fields

AWK breaks lines into fields

By default, fields are separated by whitespace, e.g

Mike	leprechaun	7	415	201
\$1	\$2	\$3	\$4	\$5

A field can be accessed by prefixing '\$' to the field number, e.g. \$2 is 'leprechaun', \$3 is '7'

```
<in> | awk '$3 == 7' # print if 3rd field equals 7
```

Comparison Operators

<code>~ !~</code>	Regular expression match and negation
<code>== !=</code>	Equals/not equals (don't use '=')
<code>< > >= <=</code>	Numeric comparisons
<code>/a/,/b/</code>	TRUE between matches (like in sed)
<code> </code>	Logical OR
<code>&&</code>	Logical AND
<code>!</code>	Logical NOT

Conditional examples (1)

print where 3rd field matches expression

```
$ awk '$3 ~ /AT1G[0-9]{5}/'
```

print where 3rd field is an integer

```
$ awk '$3 !~ /-?[0-9]+/'
```

print lines where 2nd column equals 6

```
$ awk '$2 == 6'
```

Conditional examples (2)

if no field is given, search whole line

```
$ awk '/>/'
```

can use ranges like in sed

```
$ awk '/hi/,/bye/'
```

scientific notation is OK

```
$ awk '$5 < 1e-6'
```

Conditional examples (3)

```
$ awk '$1 > 50 && $4 < 1e-3' # both true
```

```
$ awk '$6 > .5 || $2 < 1e-6' # either true
```

```
# group conditionals with parentheses
```

```
$ awk '!/^#/ && ($2 > 7 || $3 == "VIP")'
```

- AWK uses extended regular expressions

see for yourself (1)

Don't worry about the print for now ...

```
$ awk '$1 < $2 {print "TRUE"}'
```

```
$ awk '$1 ~ $2 && $3 < $4 {print "TRUE"}'
```

Can \$2 be a regular expression? Other rules?

```
$ awk '$1 ~ $2 {print "TRUE"}'
```

Resetting Field Separator

You may reset the separator with option (-F)

```
# set field separator to comma
```

```
$ awk -F, '/waldo/'
```

```
# or to TAB
```

```
$ awk -F$'\t' '/waldo/'
```

AWK builtin variables (1)

AWK has several special, builtin variables

NR - current line number

Conditional examples (2)

like `head -5` or `sed 1,5`

\$ awk 'NR == 1, NR == 5' a.txt

Print lines 1,2,5,6,9,10,...

\$ awk 'NR % 4 == 1 || NR % 4 == 2' a.txt

fastq to fasta converter

\$ awk 'NR % 4 ~ /[12]/' a.fq | **tr** '@' '>'

AWK Gotcha's

- ❖ code **MUST** be wrapped in single quotes, this passes the text as-is to the awk interpreter
- ❖ within-code strings should be double quoted, otherwise they will be interpreted as variables (unless they are in a regular expression)

Procedures

Syntax

```
condition { procedure }
```

When condition is TRUE, do procedure
(implicit IF statements)

```
$3 == "Fred" { print $2 }
```

print command

```
awk '{print $2, $1}'
```

- Prints 2nd and 1st fields
- Commas are special, they are standins for the Output Field Separator string (OFS)
- Procedures can be used alone (do all lines)
- '{' and '}' are **NOT** optional

Print Example (1)

Print 2nd and 1st fields, separated by OFS

\$ **awk** '{print \$2, \$1}'

The equivalent operation in sed

\$ **sed** -r 's/([^]+) ([^]+).*/\2 \1/'

Mathematical Operators

AWK will interpret variables as numbers if you perform mathematical operations on them.

<code>+</code>	<code>-</code>	<code>*</code>	<code>/</code>	normal plus, minus, times, div
<code>%</code>				returns remainder after division
<code>^</code>	<code>**</code>			exponentiation

Printing/Math examples

```
echo '1.14' | awk 'print $1, $2, $1 + $2'
```

```
1.145.1
```

```
echo '28' | awk 'print $1 ** $2'
```

```
128
```

```
echo '125' | awk 'print ($1 + $2) ** $3'
```

```
243
```


see for yourself (2)

Try out AWK mathematical function:

```
awk '{print $1 + $2}'
```

```
awk '{print $1 + ($2 ** $3)}'
```

etc.

How does AWK handle non-numeric fields?

Non-integers? Very large numbers?

String concatenation

- Adjacent strings are concatenated
- Spaces are ignored
- Mathematical operations have precedence over string concatenation

```
$ awk '{print $1"+"& $2 "=" $1+$2}' <<< "1 5"
```

1+5=6

Within procedure logic (1)

No implicit IF within a procedure:

FAILS!!! Syntax error

```
awk '/A/ {$1 > 5 {print "hi"}  
      $1 <= 5 {print "low"}}'
```

#

doesn't work

dies, fails

Instead, use if clause

```
awk '/A/ {if ($1 > 5) {print "hi"}  
      else {print "low"}}'
```

AWK Language Structure

AWK Structure

AWK scripts have three pieces:

1. **Beginning:** code run before text processing
2. **Middle:** actions performed on each record
3. **End:** code run after last text processing

Here we move beyond one-liners ...

BEGIN {

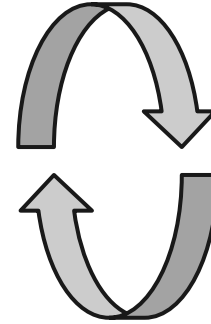
Initialize variables
Input agnostic printing

}

Data enters

condition { procedure }
condition { procedure }
condition { procedure }
condition { procedure }

Once for
each record



END {

Print any final output

}

Beginning (1)

Syntax: BEGIN{ lines of code }

<in> | awk 'BEGIN{ print "hello world" }'

This command completely ignores the input

AWK builtin variables (2)

FS - input field separator (space by default)

OFS - output field separator (OFS = FS by default)

RS - record separator (\n by default)

ORS - output record separator (ORS = RS by default)

FILENAME - the name of the current input file

Beginning (2)

Variables can be initialized in BEGIN

```
BEGIN{  
    FS="\t"  
    OFS=","  
}
```

OR (AWK isn't picky about spacing):

```
BEGIN{FS="\t"; OFS=","}
```

Middle

The middle code is run on each record of input

Syntax:

```
BEGIN { begin code }
```

```
condition1 { action1 }
```

```
condition2 { action2 }
```

```
END { end code }
```

Middle example

```
# Convert comma delimited to TAB delimited  
awk 'BEGIN {FS=","; OFS="\t"} {print}'
```

End

End is run only after all lines of output have been parsed

Prints the number of lines in the input

```
<in> | awk 'END { print NR }'
```

AWK script (1)

```
BEGIN{ print "Here are the headers!" }
```

```
/^>/ { print }
```

```
END{ print "That's all" }
```

Functions

Substitution Functions

replace every pattern in field

`gsub(pattern, replacement, field)`

replace the first pattern in the field

`sub(pattern, replacement, field)`

do not change field, return a string

`s = gensub(pattern, replacement, field)`

Substitution Examples

Remove numbers after decimal in field 2

```
awk '{ sub("\.[0-9]+$", "", $2); print }'
```

all numbers to '*' in all fields

```
awk '{ gsub("[0-9]", "*", $0); print }'
```


String Functions

<code>length(s)</code>	number of characters in <code>s</code>
<code>tolower(s)</code>	convert to lowercase
<code>toupper(s)</code>	convert to uppercase
<code>split(s, a, d)</code>	split <code>s</code> into array
<code>\$ awk '{split(\$1, a, ","); print a[2]}'</code>	
<code>substr(s, b, l)</code>	get substring
<code>\$ awk '{print substring(\$1, 3, 5)}'</code>	

see for yourself (3)

```
awk '{sub($1, $2, $3); print}'
```

```
awk '{print substr($1, $2, $3)}'
```

```
awk '{print toupper($1),  
      tolower($1),  
      length($1)}'
```

Math Functions

`exp(x)`, `log(x)` # e^x and $\ln x$

`int(x)` # cuts after decimal point

`sqrt(x)` # square root of x

`rand(x)` # returns random number $[0,1]$

`srand(x)` # resets random seed

`cos(x)`, `sin(x)`, `atan2(y,x)`

Math function examples

Take the log of the 5th column (print all)

```
awk '{sub(".*", log($5), $5); print}'
```

Replace integer in 4th column with random integer
between 0 and 999

```
awk 'BEGIN{ srand() }  
      {sub("[0-9]{3}", int(rand() * 1e3), $4); print}'
```

To tap the local entropy pool and get safer random seeds, try this:

```
awk -v s=$RANDOM$RANDOM 'BEGIN{ srand(s) }  
      {sub("[0-9]{3}", int(rand() * 1e3), $4); print}'
```

see for yourself (4)

```
awk '{print "e^x=" exp($1), "log(x)=" log($1)}'
```

```
awk 'BEGIN{srand()} {print rand()}'
```

Your Functions

```
pi = 4 * atan2(1,1)
# Box-Muller transform: produces two normal random variables
function rnorm(pi, a, b){
  r1 = rand(); r2 = rand() # all variables are global
  a = sqrt(-2 * log(r1)) * cos(2 * pi * r2)
  b = sqrt(-2 * log(r1)) * sin(2 * pi * r2)
  return # return takes no arguments
}
{rnorm(pi, a, b); print a "\n" b}
```

When NOT to use AWK

Data that are structured in a more complicated manner, need specialized tools

e.g. XML, HTML, ASN.1, csv files when fields contain delimiters

Supplementary Material

S.M. 1

Variables, arrays and for-loops

Variable Operators

`x = 10.5` # give x value 10.5

`x *= 2` # multiply and reassign

`x += 5` # add five to variable x

`x++ x--` # increment/decrement

also: `/=`, `^=`, `-=`

Uninitialized variables treated like 0 or “

AWK Variables

Calculate mean of column of numbers

```
awk '{ a += $1 } END { print a / NR }'
```

a search pattern can be a variable

```
awk 'BEGIN{s="Ra1ph"}  
      $1 == s { s = $2}  
      END{ print s }'
```

AWK Arrays (1)

AWK has associative arrays, like Perl hashes or Python dictionaries.

They map a key to a value.

`a[key] = value` # map a key to a value

`a[key] *= 2` # multiply value of 'key' by 2

AWK for-loop

C-style for-loop

for (i=0; i < NF; i++) { **procedure** }

by key in array

for (k in a) { **procedure** }

Array Examples

```
awk '{a[$3] += $4}  
      END{for(k in a) {print k, k[a]}}'
```

Passing variables with -v

```
awk -v x=5 'BEGIN{ print x }'
```

```
awk -v seed=$RANDOM 'BEGIN{srand(seed)}  
{print $1 + (rand() - rand()) * ($1 / 100)}
```