Part Four

AWK: columnar data and mathematical functions

What is AWK?

- AWK is a full programing language
    - variables and arrays (like Perl hashes)
    - loops and conditional statements
- Sed-like addressing and regular expressions
- Automatically splits lines into words
- Optimized for text parsing

Terms

- **record** - usually a line of input
- **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

- Condition statements
  - condition only calls
  - fields and conditional logic
  - field separator
- Procedure statements
  - print
  - mathematical operators

1. Condition Statements

Condition only calls

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

Sample Data

Navigate to **`4th`** folder

`a.tab - adapted from m.tab in 3rd`

DYI (1)

```
awk ‘/Feb/’ a.tab
awk ‘/Apr/,/Sep/’ a.tab
awk ‘1,5’ a.tab  # fyi doesn’t work
awk ‘/Jan/’ a.tab
awk ‘/[RB]ob/’ a.tab
```

AWK Fields

AWK breaks lines into fields

By default, fields are separated by whitespaces, 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’

```
awk ‘$3 == 7’ a.txt # print if 3rd field equals 7
```

Comparison Operators (1)

~	Regular expression match
!~	Regular expression non-match
==	Equals (don’t use ‘=’)
!=	Not equals
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to
/a/,/b/	TRUE between matches (like in sed)

DYI (2)

Now we can test again a single column

```
$ awk ‘$4 ~ /Jan/’
$ awk ‘$1 !~ /^-?[0-9]+$/’
$ awk ‘$3 < 1e-6’
```

Logical Operators

`||`	Logical OR
`&&`	Logical AND
`!`	Logical NOT

These are used to string conditions together

```
(<condition1> || <condition2>) && ! <condition3>
```

Conditional examples

```
$ 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”)’
```

DIY (3)

```
# Don’t worry about the print for now ...
$ awk ‘$1 < $2 {print “TRUE”}’
# Can $2 be a regular expression?
$ awk ‘$1 ~ $2 {print “TRUE”}’
# Does this work for numbers and strings?
$ 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)

```
# print the 5th line
$ awk ‘NR == 5’ a.tab
# like `head -5` or `sed 1,5`
$ awk ‘NR == 1, NR == 5’ a.tab
```

```
# fastq to fasta converter
$ awk 'NR % 4 ~ /[12]/' a.fq | tr ‘@’ ‘>’
```

Warning about quotes

```
awk “{print $1}” # WRONG
```

Here AWK gets the *shell variable* $1 instead of a literal string ‘$1’

This shell variable, will usually be undefined

Procedures

Syntax

condition { **procedure** }

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

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

print command

```
awk ‘{print $2, $1}’
```

- Prints 2nd and 1st fields
- Commas are special, they are field separators
- Procedures can be used alone
- ‘{‘ and ‘}’ are **NOT** optional

Comparison to sed

Problem: Print 2nd and 1st fields of input

# solution in awk

```
$ awk ‘{print $2, $1}’
```

# solution in sed

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

Mathematical Operators

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

`+ - * /`	`normal plus, minus, times, div`
`^ **`	`exponentiation`
`%`	`modulo operator - returns remainder after division`

Math examples

```
echo ‘1.1 4’ | awk ‘print $1, $2, $1 + $2’
1.1 4 5.1
echo ‘2 8’ | awk ‘print $1 ** $2’
128
echo ‘1 2 5’ | awk ‘print ($1 + $2) ** $3’
243
```

String concatenation

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

```
$ echo “1 5” | awk ‘{print $1“+”$2 “=” $1+$2}
1+5=6
```

DIY (4)

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?

AWK Variables

On each line, add $1 to **x**

```
awk ‘{x = x + $1} END {print x}’
```

At the end, print **x**

Prints the sum of column 1

AWK Arrays

Add $1 to the $2 array category

```
awk ‘{a[$2] += $1}
      END{ for(v in a){ print v, a[v] } }’
```

For each $2 category, print the $1 sums

DIY (5)

Write an awk command to sum a column

Write a command to sum $1 across $2 in *a.tab*

AWK as a language

```
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}
```