Part Three

Scripting in AWK

What is AWK?

- AWK is a full programing 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

Outline

1. Condition statements
2. Procedure statements
3. Language structure
4. Supplementary Material

1. Condition Statements

Condition only calls

AWK Rule 1: If the commands consists only of a condition, print each record for which that condition returns TRUE

Simple Examples

```
# Print any lines containing ‘Here’ (like sed)
$ awk ‘/Here/’
# Print from ‘Here’ to ‘There’ (also like sed)
$ 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` holds ‘may’, `$3` is ‘not’

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

Comparison Operators

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

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 ‘@’ ‘>’
```

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 interpet variables as numbers if you perform mathematical operations on them.

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

Printing/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

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

- Mathematical operations have precedence over string concatenation

Within procedure logic (1)

No implicit IF within a procedure:

```
# FAILS!!! Syntax error                     #
awk ‘/A/ {$1 > 5 {print “hi”}               # die die
          $1 <= 5 {print “low”}}’           # die die
# Instead, use if clause
awk ‘/A/ {if ($1 > 5) {print “hi”}
          else {print “low”}}’
```

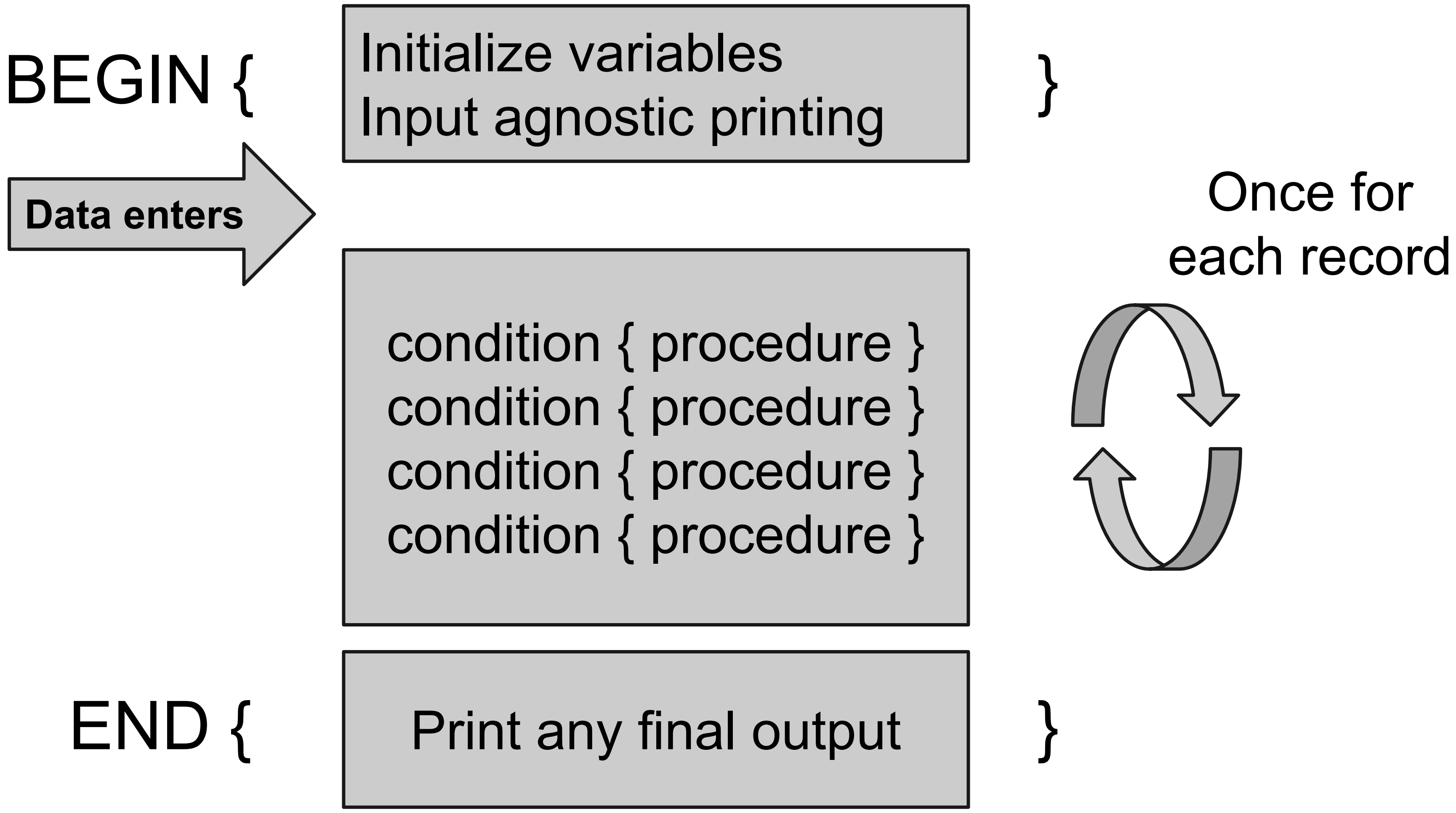
AWK Language Structure

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
Once for each record
condition { procedure }
condition { procedure }
condition { procedure }
condition { procedure }
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 be 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

```
length(s)          number of characters in s
tolower(s)         convert to lowercase
toupper(s)         convert to uppercase
split(s, a, d)  split s into array
   $ awk ‘{split($1, a, “,”); print a[2]}’
substr(s, b, l) get substring
   $ awk ‘{print substring($1, 3, 5)}’
```

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}’
```

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 reasign
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=”Ralph”}
     $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)}
```

S.M. 3
Input/Output