(g)awk 1

Lecture 13

Michael J Wise

Awk

- Computer program names, particularly in the Unix world, are often puns or acronyms; Awk is both.

BBC
https://www.bbc.com/news/science-environment-50563953

Great Auk hunted to extinction C19

Awk is the initials of the 3 people who created hit: Aho, Weinberger and Kernighan. Early Awk books had pictures of an Auk

Diversity of Awks

- Multiple different implementations of AWK
    - *AWK – original version from Bell Labs (1977)*
    - *Gawk – GNU AWK (1988).*
    - *Nawk – New AWK (1993)*
- Calling awk can end up invoking any of these, or the even newer Mawk (found in Ubuntu).
    - *System dependent*
- We will be using Gawk in this unit as it's the most widely used.

Awk

- Somewhat like Sed, Awk can be viewed as set of pattern-action rules which are applied to lines from an input file

- Like Sed, Awk rules can be specified on the command-line or (generally) via a file of rules – *awk script.*

- Awk syntax based on C (Kernighan was involved in creating an early standard K&R C)

- The basic information unit for Sed is the input line; for Awk it is the fields within the input line

Rules and Fields

- A rule is one or more Awk statements enclosed in parentheses { }. The rule can be preceded by a pattern

- A rule without a pattern is applied to all input lines

- awk regards input lines as a sequence of fields separated by a field separator character.
  - *By default this is <tab> or <blank>.*

- `$0` refers to the entire line. → *This is different from shell argument*

- `$1` refers to the first field, `$2` to the second, etc.

```
gawk '{print $0}' test_file
```

- *prints the contents of `test_file` on stdout*

- *Note the use single-quotes to protect the gawk statement from the Shell.*

Gawk commands

- Gawk has relatively few command-line options:
    - *-F* *<Char> – set Char to be the field separator*
    - *-f <file> - take the rules from the file*

    - *-v <variable>=<value> create a variable and give it a value at the command line. The variable (with that value) will be available to the Awk script*

Example

```
gawk -F # '{print $2 "#" $1}' test_file
```

- sets # as the field-separator, and for each input line prints the second field, followed by #, followed by the first field.

Test_file

```
Mozart#WA#8640352
Orff#C#8777251
Brahms#J#7531430
Vaughan-Williams#R#8707067
Saint Saens#C#6940827
```

Stdout

WA#Mozart
C#Orff
J#Brahms
R#Vaughan-Williams
C#Saint Saens

Awk variables

- Awk variables appear as required (like Python)

- An Awk variable's type depends on usage. String is default, with initial value `""`, but will be interpreted as `0` if appropriate (unlike Python)

- The format for an assignment to a variable is:

<variable> = *<expression>*

- To use a variable in an expression, just use the name (no `$` required!), e.g.

```
x = 42 # The type of x is now integer
```

Built-in Awk variables

- Like Shell, Awk has a number of built-in variables that can be very handy. Here are some

ARGC	Command-line argument count
ARGV	Array of command-line arguments
FILENAME	Name of input file (there may be several)
FNR	Index of line in current file
FS	Input field separator (default blank, tab)
NF	Number of fields in the current line
NR	Index of current line (from start of first file)
OFS	Output field separator (default blank)

Mathematical expressions

- Mathematical expressions can include:
  - *+, -, *, /, % (modulus, remainder), ˆ (exponentiation).*
  - *++<variable>   (pre-increment)*
  - *<variable>++   (post-increment).*

- With pre-increment, the variable is incremented and the value returned is the new value;

- With post-increment, the value returned is the old value. The variable is then incremented.

- For example:

```
% echo 4 | awk ’{a = $1; print ++a " " a++ " "a}’
5 5 6
```

- Some maths functions

Assignment

- Like C, assignment = returns a value, so

`a = b = c = 42` understood as `a = (b = (c = 42))`

- Compound Assignment: The arithmetic operators can be combined with =, i.e. `+=`, `-=`, `*=`, `/=`, `%=` and `^=`.

- In general:

  <variable> <operator>= <expression>

  is a shorthand for:

  <variable> = <variable> <operator> <expression>

- For example:

`gawk '{x = $1; x += 42; print "With meaning of life = " x}'`

- Note single quotes, and `;` statement separator

Print

- Printing values can be done in two ways; The simplest is to use `print` (as in the examples above)

- The format for print is:

`print` *[<expression list>] [> <expression>]*
`print` *[<expression list>] [| <expression>]*

- If no expressions are provided `$0` is printed

- The optional *> <expression>* allows you specify an output file into which the output will be redirected.

- *<expression>* must evaluate to a string.

Print

- For example, the shell-script `copy_file`:

```
#!/usr/bin/env bash
gawk "{print \$0 > \"$2\"}" $1
```

Example of it being used:

```
% ./copy_file Alice_in_Wonderland.txt fred
-rw-r--r--   1 stud stud 160785 Mar  1 07:43 fred
-rwxr-xr-x 1 stud stud     52 Mar  1 07:40 copy_file
-rw-r--r--   1 stud stud 160785 Jan 18 06:25
Alice_in_Wonderland.txt
```

$1

$2

- After the variable substitutions are done, what is the Gawk command that gets executed?

Print

- >> can be used in place of >

- Alternatively, Awk allows the output to be redirected into a UNIX command via a pipe internal to the awk-script. The command is a string.

- For example, the small student database:

```
Mozart#WA#8640352#99
Orff#C#8777251#80
Brahms#J#7531430#90
Vaughan-Williams#R#8707067#85
Saint Saens#C#6940827#85
```

Print

- This script calls Awk to print all the students by decreasing mark.

```
#!/usr/bin/env bash
gawk -F"#" \
    '{print $2 "#" $1 "#" $4 | "sort -t# -k 3nr"}'
marks\
    | sed -e 's/#/ /' -e 's/#/: /'
```

- Note the use of single quotes

- The call to sort could also be done from the shell, rather than from within Awk

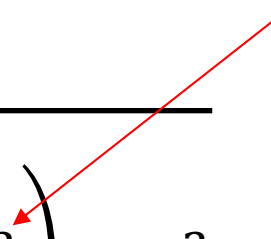In computing, as in life, punctuation matters

Two special rules

- By default (i.e. unless you have a prior condition), each set of actions within their respective { } apply to all input lines. Then depends on condition.

- Two special Rules:

  - *BEGIN { <actions> } is executed before the first line is read*

  - *END { <actions> } is executed after the last input line has been read*

Demo

- Create an awk script, `simple_avcol.awk`, to computer the mean and standard deviation of a file of numbers (one per line).

- Recall that:

$$\bar{x} = \frac{\sum_i^n i}{n}$$

Sum of squares, right?

$$S = \sqrt{\left(\frac{1}{n}\sum_i^n i^2\right) - \bar{x}^2}$$

Compute sum_x and sum_x_sqr as you go

Printf
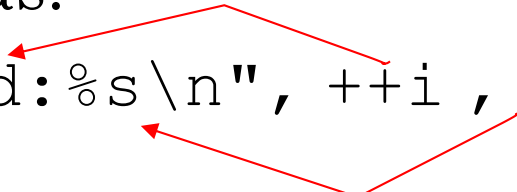
- `printf` allows the user to specify the format in which output will appear.
- Calls to `printf` look like:

  `printf(`*<format string>*`[ ,` *<expression list>]* `)`

- `printf` sends characters of the format string verbatim to stdout, except for format conversions.
- Format conversions are place-holders in the string.
- For example:

```
awk '{print ++i ": " $0}' test_file
```

- can be rewritten as:

```
awk '{printf("%d:%s\n", ++i , $0)}' test_file
```

Format Conversions

- The schema for format conversions (somewhat simplified) is:

%[<output width> [.<precision>]]<format letter>

- For example, `%d %5s %0.2f`

- The *<format letter>* of a format conversion indicates how the corresponding value will be printed.

- Values corresponding to format conversions are obtained by evaluating elements of the *<expression list>*.

- There should be as many items in the list as there are format conversions.

Formats

Format Letter	Description
c	Single Character
s	String (default: length of string)
d	Integer (default: length of integer)
e	Scientific notation (default 6 decimal places)
f	Floating point (default 6 decimal places)
g	Either e or f whichever is shorter with trailing 0 removed
%	The % itself

Notes

- If a `\n` is required, it must be explicitly added; `\t` is the <TAB> character.

- The most common formats, `%s` and `%d` do not usually require the output width to be specified.

- Floating point reals and scientific notation reals usually require the field width to be specified, e.g.

```
printf("Min: %d, Max: %df\n",min, max)
printf("Mean: %0.3f", sum/n)
printf("\tN: %d\n",n)
```

- How many lines were printed?