
THE UNIVERSITY OF WESTERN AUSTRALIA

# (g)awk 2

## Lecture 14

Michael J Wise

# Conditional Rule Execution

- It also very common to prefix a set of actions with a test, which limits the application of the actions to those input lines that satisfy the test.
- BEGIN and END are particular cases.
- For example:

```
gawk -F"#" '$3~/^8/ {print}' test_file
```

- prints only the lines whose third field begins with an 8. ~ is called a matchop. (See later.)
- In general, the format for a rule is:

*[<test>]* { *<statements>* }  - similar to Sed

# Conditional Rule Execution

- One format for tests is:

*<expression> <relop> <expression>*

- The possible relational-operators are: ==, !=, <, <=, >, >=.
- Watch out that you don't use = in place of == !
- Example
- If `test_lines.awk` contains:

```
NF != 4 {print NF " fields in line: " NR}
```

- Then

```
gawk -F ",”-f test_lines.awk test_file.csv
```

- Reports all the lines that do not contain 4 fields
  - *Data cleaning*

# Matchop

- A second format for tests is:

*<expression> <matchop> <regular expression>*

- The two regular-expression matching operators are:

`~` (match) and `!~` (does not match). The regular expression is contained between `/ /`.

- For example, the following shell script counts the number of blank lines in the input file:

```
#!/usr/bin/env bash
gawk '$0 ~ /^$/{sum++}END{print sum}' $1
```

- Notice that there are three different uses of the `$` symbol! Note also the single quote

```
gawk '$0 == "" {sum++}END{print sum}' $1
```

# What is *True*

- In Awk (and C), the value *FALSE* is indicated by the integer value `0`.
- The value *TRUE* is ANY integer value other than `0`, but typically `1`.
- Opposite to Shell!!!

# Combining Tests

- Tests, e.g involving <*relop*> or <*matchop*>, can be combined.

<*test*> && <*test*>

- evaluates to TRUE if both of the components are TRUE; FALSE otherwise.

<*test*> || <*test*>

- evaluates to FALSE if both of the components are FALSE; TRUE otherwise.

! <*test*>

- inverts the sense of <test>.
- Note: The formats shown here are only in terms of two tests. There can be more, but you might have to use bracketing to make the meaning clear.

# If .. Else

- Test before a set of actions is a fairly blunt instrument; the block of statements happen, or they don't happen.
- Sometimes also want to take alternate actions within a rule

# If .. Else

- The format for the `if .. else` statement is:

`if` ( *<condition>* )

*<statement or statement block>*

*[*`else`

*<statement or statement block> ]*

- If the *<condition>* evaluates to TRUE (i.e. nonzero)

the *<statement or statement block>* is executed; otherwise, the *<statement or statement block>* following the `else` – assuming it exists!

- A statement block is simply a sequence of statements, which must be enclosed in parentheses.

# If .. Else

- If only one statement appears, it does not need to be enclosed in parentheses.
- The `else` part can, of course, be another `if` statement, but there is NO equivalent to Shell `elif`

# Demo

- Back at `simple_avcol.awk` in L12,
  - *What if `n==0` ?*
    - Test at the end, right?
  - *What if we also want the maximum and minimum values in the list?*
    - Can be done as we go
- ==> The next cut, `less_simple_avcol.awk`

# Arrays

- Like Awk variables, Awk arrays appear as required.
- The number of elements in the array is not declared. Elements appear as required.
- To assign to an array:

*<array name>*`[`*<array subscript>*`]` = *<value>*

- For example:

```
{lines[++i] = $0}
```

- assigns successive input lines to the array `lines`, subscripted by line numbers.

```
{lines[NR] = $0}
```

- does the same thing.

# Array Subscripts and Values

- Array subscripts can be any value, including strings. Provides an associative memory, like Python dictionaries
- For example, with the student database, `test_file`:

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

- there could be two arrays created in a rule:

```
{surname[$3] = $1;
initials[$3] = $2;
...
}
```

- Each array is indexed by the third (SID) field.

# Deleting Arrays and Elements

- If you want to reset an array element you can simply assign "" to the element: `a[10] = ""`
- Alternatively, you can `delete` the element:

```
delete a[10]
```

removes it entirely. Saves space, which can be important if the array becomes large

- Can also delete entire arrays

# Test for Inclusion in an Array

- Awk provides a test of whether an item is in an array.

<*expression*> `in` <*array name*>

- Evaluates to TRUE if the value of the <*expression*> is a subscript of the array <*array name*>.
- That is:

```
if(6940827 in surname)
```

...

- is the same as saying:

```
if(surname[6940827] != "")
```

# Traversing an Array

- If you are using an array, you will often want to list all the elements.
- Listing elements could be problematic in Awk because any string can be a subscript.
- Awk provides a special kind of `for` loop.

`for(`<*variable*>` in `<*array name*>`)`

<*statement or statement block*>

- For example:

```
for(i in surname)
   print initials[i] " " surname[i]
```

- NOTE: With this for loop the elements of the array will appear in *random* (though fixed) order.

# Example

- The Awk script, `hist_randints.awk`, counts the frequencies of occurrence of integers in a list.

```
{ hist[$1]++; }
END{
   for(i in hist)
       print i ":" hist[i]
}
```

- Only problem is that Awk lacks a sort function, so while ascending integer indices will generally be printed in ascending order, you can't rely on it
  - *Pipe to sort (i.e. external to Awk) possible work-around*

# Demo

- The earlier version of `avcol` used an on-the-fly method for computing standard deviation which makes some assumptions.
- Lets redo `avcol` using the usual definition of the standard deviation calculation

$$s = \sqrt{\frac{1}{n}\sum_{i}^{n}(x_i - \bar{x})^2}$$

- Use the main loop to compute the mean and store values in an array
- Use a for loop in END to compute SD