Scripting & Computer Environments

Advanced Filters II

IIIT-H

Aug 17, 2013

...Previously & Today...

Previously:

- Regular Expression (Regex) basics
- 2 Regex-Aware Filters
  1. `grep`
  2. `sed`

Today:

1. `sed` Revisited
2. `awk` Basics

...Previously & Today...

Previously:

- Regular Expression (Regex) basics
- 2 Regex-Aware Filters
  1. `grep`
  2. `sed`

Today:

1. `sed` Revisited
2. `awk` Basics

Brainstorm

1. Shell vs Regex metacharacters?

2. Regex for?

```
heard, hard, herd

cool, coolant, cooler, coolest, coolness
```

3. Decode?

```
sed -n "1,/^$/p"

sed "/^ya*y/,/[0-9]$/d"

sed -n 's/four/char/gpw output.txt' hinglish.txt
```

Brainstorm

1. Shell vs Regex metacharacters?
2. Regex for?

   `heard, hard, herd`

   `cool, coolant, cooler, coolest, coolness`

3. Decode?

```
sed -n "1,/^$/p"
```

```
sed "/^ya*y/,/[0-9]$/d"
```

```
sed -n 's/four/char/gpw output.txt' hinglish.txt
```

Brainstorm

1. Shell vs Regex metacharacters?
2. Regex for?

```
heard, hard, herd

cool, coolant, cooler, coolest, coolness
```

3. Decode?

```
sed -n "1,/^$/p"

sed "/^ya*y/,/[0-9]$/d"

sed -n ‘s/four/char/gpw output.txt’ hinglish.txt
```

Pros & Cons

Some Pros:

- Regex handling
- Search and replace feature
- Fast

Some cons:

- No feature for numeric computation
- Going backward in the file not possible

The `Awk` Filter

- Named after its authors: **A**lfred Aho, Peter **W**einberger, and Brian **K**ernighan.
- A powerful *programming language* for text manipulation + report writing (precursor to `perl`).
- C-like syntax (functions, arrays, `if`, `for` & `while` constructs, etc).
- Combines features from many filters (e.g. `grep`, `sed`).
- Flavors: `new awk (nawk)`, `GNU awk (gawk)`, `...`

`Awk`: Salient Features

- Processes a line at a time (like `sed`)
- Numeric processing
- Can manipulate *fields* of a line (N.B. `sed` processes lines)
- Regex-aware (ERE)
- Report formatting capabilities
- C-like. The implication?

Awk Usage

```
awk [options] ‘pattern {action}’ file(s)
```

- Searches for `<pattern>` and applies `<action>` on it.
- Default action is to print current record on `STDOUT`.
- Default pattern is to match all lines.
- If file(s) not specified, input taken from??
- Common options:
  - `-f` read program/pattern from a file
  - `-F` sets field separator (FS) value (default is “ ”)

In `Awk`,

- Each line in the file ≡ record ($0)
- Each column ≡ field. ($1, $2, $3, ...)

Example

```
ls -l | awk '{print}'

ls -l | awk '{print $0}'          (How about $1, $2 ...?)

ls -l | awk '/^d/ {print $1,$8}'          (The comma??)

ls -l | awk '$5>100 {print $8}'

awk '/Sa[mt]r*/' file.txt

awk -F: '{print $7}' /etc/passwd
```

print vs printf

- Both write to STDOUT
- (un)?formatted output
- The C-like `printf` takes format specifiers (`%d`,`%f`,`%s`)

```
awk '{ print $1, $2, $3) }' sales.txt
```

```
awk '{ printf("%6s %4d %-8f \n", $1, $2, $3) }' sales.txt
```

(The '-' symbol left-justifies)

Operators

Logical

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

Arithmetic

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulo
$\wedge$	Exponentiation

Relational

`<`, `<=`	less, less or equal
`>`, `>=`	greater, greater or equal
`==`, `!=`	equal to, not equal to
$\sim$, `!`$\sim$	for regex comparison

Example

```
echo 100 8 | awk '{print $1 ^ $2}'

ls -l | awk '$2==2 {print}'

awk '$2 * $3 > 50 {print}' sales.txt

awk '$3 > 10 && $4 > 20 {print}' sales.txt

awk -F: '$1 ~ /^root/ {print}' /etc/passwd
```

Variables

- No primitive data types (**char, int, float ...**)
- Either **string** or **number** (implicitly set to " " and 0 resp)
- Built-in + user-defined variables (no need to declare them)

FS	Field separator (default is space/tab)
RS	Record separator (default is newline)
NF	# of fields in the current line
NR	# of lines read so far
FILENAME	Name of the current file

```
ls -l | awk '{print $1, NF}'

awk -F, 'NR==2, NR==10 {print NR, $1}' file.csv
```

BEGIN...END

```
BEGIN {action}
END {action}
```

- These optional sections are for pre- and post-processing work.
- Way of telling `Awk` to do something before and after scanning through the file.
- Example usage:
  - `BEGIN`: generate report header, initialize variables, etc
  - `END`: print final result of computation, print output status, etc

Example

```
awk ‘BEGIN {n=1} {print $0, n++} END {print "Bye"}’ file.txt

ls -l | awk ‘BEGIN {printf "Permissions \t File Name \n"}
            { printf "%s \t %s \n", $1, $8 } ’

ls | awk ‘BEGIN { print "List of C files:" } /\.c$/ {print}
          END { print "Done!" } ’
```

The if Statement

- Awk provides control flow statements:

  Branching (if...else) + loop (for, while & do...while).

if...else

```
{ if (condition) {statment 1} else {statement 2} }
```

Example

```
ls -l | awk '$5 > 1000 { print }'


ls -l | awk '{ if ($5 > 1000) { print "Above the threshold"}
                else { print "Below the threshold" } } '
```

The if Statement

- Awk provides control flow statements:

  Branching (if...else) + loop (for, while & do...while).

if...else

```
{ if (condition) {statment 1} else {statement 2} }
```

Example

```
ls -l | awk ‘$5 > 1000 { print }’


ls -l | awk ‘{ if ($5 > 1000) { print "Above the threshold"}
             else { print "Below the threshold" } } ’
```

Awk In Action: Examples

1. Swap the order of any two columns of a file.
   ```
   awk ‘{print $2, $1} < input.txt > output1.txt
   ```
2. Delete the 3rd column of `ls -l`.
   ```
   ls -l | awk ‘{$3 = ""; print}’ > output2.txt
   ```
3. Find the maximum/minimum value of a column.
   ```
   awk ‘BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}’
   ```
4. Find the average of a column of data.
   ```
   cat input.txt | awk ‘BEGIN {ave=0} {ave+=$1} END {print ave/NR}’
   ```
5. Calculate the sum of all columns of data.
   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.

```
awk '{print $2, $1} < input.txt > output1.txt
```

2. Delete the 3rd column of `ls -l`.

```
ls -l | awk '{$3 = ""; print}' > output2.txt
```

3. Find the maximum/minimum value of a column.

```
awk 'BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}'
```

4. Find the average of a column of data.

```
cat input.txt | awk 'BEGIN {ave=0} {ave+=$1} END {print ave/NR}'
```

5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.

```
awk '{print $2, $1} < input.txt > output1.txt
```

2. Delete the 3rd column of `ls -l`.

```
ls -l | awk '{$3 = ""; print}' > output2.txt
```

3. Find the maximum/minimum value of a column.

```
awk 'BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}'
```

4. Find the average of a column of data.

```
cat input.txt | awk 'BEGIN {ave=0} {ave+=$1} END {print ave/NR}'
```

5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.
   ```
   awk ‘{print $2, $1} < input.txt > output1.txt
   ```
2. Delete the 3rd column of `ls -l`.
   ```
   ls -l | awk ‘{$3 = ""; print}’ > output2.txt
   ```
3. Find the maximum/minimum value of a column.
   ```
   awk ‘BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}’
   ```
4. Find the average of a column of data.
   ```
   cat input.txt | awk ‘BEGIN {ave=0} {ave+=$1} END {print ave/NR}’
   ```
5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.

   ```
   awk ‘{print $2, $1} < input.txt > output1.txt
   ```

2. Delete the 3rd column of `ls -l`.

   ```
   ls -l | awk ‘{$3 = ""; print}’ > output2.txt
   ```

3. Find the maximum/minimum value of a column.

   ```
   awk ‘BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}’
   ```

4. Find the average of a column of data.

   ```
   cat input.txt | awk ‘BEGIN {ave=0} {ave+=$1} END {print ave/NR}’
   ```

5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.

```
awk '{print $2, $1} < input.txt > output1.txt
```

2. Delete the 3rd column of `ls -l`.

```
ls -l | awk '{$3 = ""; print}' > output2.txt
```

3. Find the maximum/minimum value of a column.

```
awk 'BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}'
```

4. Find the average of a column of data.

```
cat input.txt | awk 'BEGIN {ave=0} {ave+=$1} END {print ave/NR}'
```

5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.
```
awk '{print $2, $1} < input.txt > output1.txt
```
2. Delete the 3rd column of `ls -l`.
```
ls -l | awk '{$3 = ""; print}' > output2.txt
```
3. Find the maximum/minimum value of a column.
```
awk 'BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}'
```
4. Find the average of a column of data.
```
cat input.txt | awk 'BEGIN {ave=0} {ave+=$1} END {print ave/NR}'
```
5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.
```
awk '{print $2, $1} < input.txt > output1.txt
```
2. Delete the 3rd column of `ls -l`.
```
ls -l | awk '{$3 = ""; print}' > output2.txt
```
3. Find the maximum/minimum value of a column.
```
awk 'BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}'
```
4. Find the average of a column of data.
```
cat input.txt | awk 'BEGIN {ave=0} {ave+=$1} END {print ave/NR}'
```
5. Calculate the sum of all columns of data.

   Read about loop statments!

Awk In Action: Examples

1. Swap the order of any two columns of a file.
```
awk ‘{print $2, $1} < input.txt > output1.txt
```
2. Delete the 3rd column of `ls -l`.
```
ls -l | awk ‘{$3 = ""; print}’ > output2.txt
```
3. Find the maximum/minimum value of a column.
```
awk ‘BEGIN {max = 0} { if ($1>max) max=$1 } END {print max}’
```
4. Find the average of a column of data.
```
cat input.txt | awk ‘BEGIN {ave=0} {ave+=$1} END {print ave/NR}’
```
5. Calculate the sum of all columns of data.

   Read about loop statments!

Final Remarks

- What has been discussed so far is just tip of the iceberg.
- `Awk`'s programming features not discussed today:
  - Loop statments (`for, while, do...while`)
  - Arrays
  - Functions
- There are many `Awk` one-liners. Check out Commandlinefu and here too.

Next...