CS2042 - Unix Tools

Fall 2010

Lecture 7

Hussam Abu-Libdeh

based on slides by David Slater

September 22, 2010

Organization

- Homework 2 due tomorrow at 11:59PM
- Any questions?

Shell shortcuts

Make entering commands easier:

- Tab completion
- Up-down arrow: browse through command history
  - so you do not have to retype everything
- Ctrl + e: jump cursor to end of line
- Ctrl + a: jump cursor to beginning of line
- Ctrl + u : delete everything from cursor to beginning of line
- Ctrl + k : delete everything from cursor to end of line
- Ctrl + l : clear the screen
- Ctrl + r : *search* the command history

More shortcuts at:
`linuxhelp.blogspot.com/2005/08/bash-shell-shortcuts.html`

gawk

- `gawk` is the GNU implementation of the `AWK` programming language
- `AWK` allows us to setup filters to handle text as easily as numbers (and much more)
- The basic structure of a `awk` program is

```
pattern1 { commands }
pattern2 { commands }
...
```

- patterns can be regular expressions! Gawk goes line by line, checking each pattern one by one and if its found, it performs the command.

AWK introduction

- `AWK` is a programming language designed for processing text-based data
  - allows us to easily operate on fields rather than full lines
  - works in a *pattern-action* matter, like `sed`
  - supports numerical types (and operations) and control flow (`if-else` statements)
  - extensively uses string types and associative arrays
- Created at Bell Labs in the 1970s
  - by Alfred Aho, Peter Weinberger, and Brian Kernighan
- An ancestor of `Perl`
  - and a cousin of `sed` :-P
- Very powerful
  - actually *Turing Complete*

Why gawk and not sed

- convenient numerical processing
- variables and control flow in the actions
- convenient way of accessing fields within lines
- flexible printing
- built-in arithmetic and string functions

Simple Examples

```
gawk '/[Mm]onster/ {print}' Frankenstein.txt
gawk '/[Mm]onster/' Frankenstein.txt
gawk '/[Mm]onster/ {print $0}' Frankenstein.txt
```

- All print lines of Frankenstein containing the word Monster or monster.
- If you do not specify an action, gawk will default to printing the line.
- $0 refers to the whole line.
- gawk understands **extended** regular expressions, so we do not need to escape +, ? etc

Begin and End

- Gawk allows blocks of code to be executed only once, at the beginning or the end.

```
gawk 'BEGIN {print "Starting search for a monster}

     /[Mm]onster/ { print; count++}

END {print "Search completed, there are " count " monsters in the book.}

' Frankenstein.txt
```

- gawk does not require variables to be initialized
- integer variables automatically initialized to 0, strings to "".

gawk and input fields

The real power of gawk is its ability to automatically separate each input line into fields, each referred to by a number.

```
gawk '
BEGIN {print "Beginning operation"; myval = 0}
/debt/ { myval -= $1}
/asset/ { myval += $1}

END { print myval}' infile
```

- $0 refers to the whole line
- $1, $2, ... $9, $(10) ... refer to each field
- The default Field Separator (FS) is white space.

gawk gawk gawk

- If no pattern is given, the code is executed for every line

```
gawk ' {print $3 }' infile
```

Prints the third field/word on every line.

Other gawk variables

- `NF` - # of fields in the current line
- `NR` - # of lines read so far

```
gawk '{for (i=1;i<=NF;i++) print $i }' infile
```

Prints all words in a file

- You **cannot** change `NF` or `NR`.

The field separator

- `FS` - The field separator
- Default is " "

```
gawk 'BEGIN { FS = ":"}
toupper($1) ~ /FOO/ {print $2 } ' infile
```

- `gawk -F:` also allows us to set the field separator
- `toupper()`, `tolower()` - built in functions
- ~ - gawk matching command
- !~ - gawk not matching command

What type of code can I use in gawk?

gawk coding is very similar to programming in c

- `for(i = ini; i <= end; increment i) {code}`
- `if (condition) {code}`
  (In both cases the { } can be removed if only one command is executed)
- and so on. See the gawk manual for more

www.gnu.org/software/gawk/manual

gawk examples

```
gawk ' {
      for(i=1;i<=NF;i++){
            for(j=length($i);j>0;j--) {
                  char = substr($i,j,1)
                  tmp = tmp char
            }
            $i = tmp
            tmp = ""

      } print
} ' infile
```

gawk examples

```
gawk ' {
      for(i=1;i<=NF;i++){
            for(j=length($i);j>0;j--) {
                  char = substr($i,j,1)
                  tmp = tmp char
            }
            $i = tmp
            tmp = ""

      } print
} ' infile
```

- Inverts all strings in the file

Variables and Associative Arrays

- gawk handles variable conversion automatically

`total = 2 + "3"` assigns 5

- Arrays are automatically created and resized
- Arrays are "associative", meaning the index can be any string:

`array["txt"] = value`
`array[50]` is equivalent to `array["50"]`.

Array functions

The following are very helpful:

```
if (someValue in theArray) {
    action to take if somevalue is in theArray
} else {
    an alternate action if it is not present
}

for (i in theArray) print i
```

Associative Array Example

Suppose we have an iou file of the following form:

```
Who owes me what as of today
Name \tab Amount
Name \tab Amount
⋮
```

Lets write a gawk script to add up how much everyone owes us

Associative Array Example

```
gawk '
     BEGIN {FS = "\t" }
     NR > 1 { Names[$1]+=$2 }
     END { for(i in Names) print i " owes me " Names[i] " Dollars."}
' ioufile
```

Matching and gawk

gawk can match any of the following pattern types:

- /regular expression/
- relational expression
- pattern && pattern
- pattern || pattern
- patern1 ? pattern2 : pattern3 - if pattern1, then match pattern2, if not then match pattern3
- (pattern) - to change order of operations
- ! pattern
- pattern1, pattern2 - match pattern1, work on everyline until it matches pattern2 (not combinable)

Last words on gawk

- We have only touched on the very basic things you can do with gawk to give you a taste
- Other Things:
  - fancy output using `printf`
  - while loops
  - built-in functions (sin, cos, length, substr, system, exit etc)