CS2043 - Unix Tools & Scripting

Cornell University, Spring 2014[1]

Instructor: Bruno Abrahao

February 12, 2014

[1] Slides evolved from previous versions by Hussam Abu-Libdeh and David Slater

A note on sed

`sed 's/regexp/string/ file'`

- process line by line in main memory
- sends output to `stdout`
- doesn't change `file`

A note on sed

Don't ever do this:

```
sed 's/regexp/string/ file' > file
```

- file is going to be replaced by a new empty file before sed starts processing it.

A note on sed

```
sed 's/regexp/string/' file > outfile
```

- process line by line in main memory
- sends output to `outfile`
- doesn't change `file`
- check `outfile` for the desired output
- then, move
  `$ mv outfile file`

A note on sed

```
sed 's/regexp/string/' file > outfile
```

- process line by line in main memory
- sends output to `outfile`
- doesn't change `file`
- check `outfile` for the desired output
- then, move
  ```
  $ mv outfile file
  ```

How do you check the output?

diff

```
diff file1 file2
```

Output

- `n{c,a,d}m`: one of line change (c), addition (a), deletion (d) occurred in line `n` of file1 compared to line `m` of file2.
- <: means that this line is exclusive of `file1`
- >: means that this line is exclusive of `file2`

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

AWK is not awkward!

- 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*

gawk

- `gawk` is the GNU implementation of the `AWK` programming language. On BSD/OS X the command is called `awk`.
- `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.

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 at the end.

```
gawk 'BEGIN {print "Starting search"}

    /[Mm]onster/ { count++}

END {print "Found " count " monsters in the book!}
' Frankenstein.txt
```

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

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.

Simulation

Let's implement `wc -l` in awk!

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 'print $N' file`

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

The field separator

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

```
gawk 'BEGIN { FS = ","} {print $2 }' infile
```

- `gawk -F:` also allows us to set the field separator

Statistics

Let's compute some CS 2043 statistics based on your survey responses using awk!

Matching and gawk

gawk can match any of the following pattern types:

- /regular expression/
- relational expression
- exp && exp
- exp || exp
- condition ? statement1 : statement2 - if condition, then statement1, else statement2
- ! exp
- and more...

Relational Operators 1/2

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

- `toupper()`, `tolower()` - built in functions
- $\sim$ - gawk matching relational operator
- $!\sim$ - gawk not matching operator

Demo

Lottery example

Other gawk functions

- `exp(x)` : exponential of `x`
- `rand()` : produces a random number between 0 and 1
- `length(x)` : returns the length of x
- `log(x)` : returns the log of x
- `sin(x)` : returns the sin of x
- `int(x)` : returns the integer part of x

Relational Operators 2/2

```
gawk '($1 > .5){print $2 }' infile
```

Other relational operators

- `<`, `<=`, `>`, `>=`, `!=`, `==`

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
```

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 where only one command is executed)

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`.

Next time

More powerful features of AWK. Stay tuned!