

CSE 303

Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010

Lecture 6 – Utilities and Shell Wrap-Up

Outline

- What we have done so far
 - Linux model
 - Shell programming
 - Regular expression
- Plan for this lecture
 - Utility 1: sed
 - Utility 2: awk
 - Another useful utility: **find** (will not cover in class)
 - Example: `find . -name "*.txt"`
 - Shell wrap-up and where to go from here

Automating File Editing

- We have learned how to automate various simple tasks involving file manipulation and program execution
- But how about:
 - Automating file editing
 - Simplifying repetitive edits to multiple files
 - Typical example: search and replace in many files
 - Writing a conversion program (HW2, Problem 2)
- Sed: simple utility program that can help us

The Sed Editor

- Sed is a non interactive editor that interprets and performs the actions in a script
- Sed is stream-oriented
 - Input comes from file or from stdin
 - Input flows through program
 - Output goes to stdout

How Sed Works

- Sed edits a file **one line at the time**
- Each line is **copied into a pattern space**
- All editing commands are then applied...
 - On the data in the pattern space
 - **One after the other, in sequence**
- Hence, original input does not change
- Possible to restrict edits to subset of lines

Command-Line Syntax

- **Method 1: One-line syntax**

```
sed [options] 'command' file(s)
```

```
sed -e 'command1' -e 'command2' file(s)
```

- **Method 2: Scriptfile (not taught in this class)**

```
sed [options] -f scriptfile file(s)
```

Search and Replace with Sed

- Simple most common use

```
sed 's/pattern/replacement/g' file
```

- Meaning: “**Replace every (longest) substring that matches *pattern* with *replacement*.**”
- Common variations for search and replace
 - Omit `g`: replaces only first match
 - `sed -n`: suppresses normal output
 - Add `p` where you normally put `g` to print the lines that match pattern

More About Substitution

- Examples

```
sed 's/a/b/g' ex1.txt
```

```
sed -n 's/a/b/2p' ex1.txt
```

```
sed -e 's/a/b/g' -e 's/b/c/g' ex1.txt
```

- Additional examples (using regexps)

```
sed 's/.*Linux.*/\&:/' ex2.txt
```

```
sed 's/.*Linux \(.*\) .*/\1:/' ex2.txt
```

- Newline note: the `\n` is not in the matched text and is (re)-added when printed

Editing Subset of Lines

- General syntax of sed commands

`[address [, address]] [!] command [arguments]`

- Delete lines 3-5: `sed '3,5 d' ex3.c`

- Delete lines that do not say SAVE

`sed '/SAVE/! d' ex3.c`

- Delete all lines that start with //

`sed '/\\/\\/ / d' ex3.c`

- Remove all lines between /* and */

`sed '/\\/*/,/*\\ / d' ex3.c`

Advanced Features

- Commands so far: substitute, print, delete
- Other commands (not shown in this class)
 - append, replace with block, insert, translate
 - branch to label
 - multi-line patterns
 - The *hold* space for fancy editing
 - Example: copy and paste lines
- Honestly... if you need these, it might be better to use Perl or Python

Awk

- Awk is a pattern-matching program for processing text files composed of records separated by some delineator
 - Default delineator: newline character
 - Records contain **fields** (default separator space)
- Usage
 - Generate a report from logs
 - Processing results from experiment

Command-Line Syntax

- Method 1: One-line syntax

```
awk [options] 'script' file(s)
```

- Useful variant

Change the field separator from space to c

```
awk -F c 'script' file(s)
```

- Method 2: Scriptfile (not taught in this class)

```
awk [options] -f scriptfile file(s)
```

Basic Functionality

- Script consists of `pattern { procedure }`
- Awk processes a file one record at the time
- For each record
 - Access fields with `$1, ... $n`
 - Number of fields: `NF`
- Example: print only last and first fields

```
awk '{print $NF " " $1}' grades.txt
```

- Example: replace grades with average

```
awk '{print $1 " " ($2+$3)/2}' grades.txt
```

Using Patterns

- Can apply procedures only to records that match a pattern. Examples:

```
awk '/Jane/{print $2}' grades.txt
```

```
awk '/Bob/, /Jane/ {print $0}' grades.txt
```

```
awk '$2 < 8 {print $0}' grades.txt
```

- **BEGIN** and **END** patterns serve to execute operations before and after processing file

Example: compute class average on hw1

```
awk '{x+=$2; i++} END { print x/i}' grades.txt
```

- Can do pattern matching on fields as well

Advanced Features

- awk is quite a powerful scripting language
- Many features not covered in this class
 - Prog. language constructs: arrays, loops
 - Defining functions
 - Fancy printing with printf
 - Some math functions: cos(), rand()
- ... although once again, if you need all this, you might want to use Perl or Python instead

Summary

- Bash scripts are powerful tools
- But they are also complex (intricate syntax)
- Lots of “tricks”
 - Typo in variable name creates a new variable: `oops=7`
 - Typo in variable use gives empty string: `ls $oops`
 - Omit subscript, get first element of array `${array}`
 - Array-out-of-bounds
 - On assignment, increases array size
 - On use, returns the empty string
 - Be careful with spaces!
 - ... and many, many more gotchas

Bottom Line

- Never do something manually if writing a script would save you time
 - Simple shell scripts can do powerful operations
 - Other utility programs help even further
- Never write a script if you need a large, robust piece of software
- Some programming languages (Python or Perl) try to give you the “best of both worlds”
- You now know two extremes that don't (Java and bash)

Java vs Bash Programming

- Shell

(+) shorter, convenient file-access, file-tests, program execution, pipes

(-) crazy quoting rules and ugly syntax

- Java

(+) cleaner, array-checking, type checking, etc.

(+) real data structures

(-) heavier weight

Where We Are and Where We're Going

- We are done with Linux, shell, & utilities
 - You should now be comfortable performing simple operations on Linux
 - After assignment 2, you should be comfortable writing simple scripts and using simple utilities
- **Where to go from here**
 - *Learn Python and/or Perl and/or Ruby*
- Next time
 - Start to learn C

Python Example

```
#!/usr/bin/python
from sys import *
import Scientific.Statistics

inFile = open(argv[1])

# Computes avg and stddev over all assignments
hw = []
for line in inFile.readlines():
    fields = line.split()
    for element in fields[1:]:
        hw.append(int(element))

print "Raw results: ", hw
avg=Scientific.Statistics.mean(hw)
stddev=Scientific.Statistics.standardDeviation(hw)
print "Avg: %.2f, stddev: %.2f" % (avg, stddev)
```

Readings

- [Linux Pocket Guide](#)
 - Section on More Powerful Manipulations (p80-81)
- Assignment 2 instructions point to online sed and awk documentation
 - But you only need to know what we covered in lecture today
 - Also: there will be no exam questions on awk