sed, awk, & perl

CS 2204
Class meeting 13

*Notes by Mir Farooq Ali and other members of the CS faculty at Virginia Tech. Copyright 2003.

sed

- Stream editor
- Originally derived from “ed line editor”
- Used primarily for non interactive operations
  * operates on data streams, hence its name
- Usage:
  * sed options ‘address action’ file(s)
  * Example: sed ’1s/’bold/BOLD/g’ foo

sed: Line Addressing

- using line numbers (like 1,3p)
- sed ‘3,4p’ foo.txt
  * ”For each line, if that line is the third through fourth line, print the line”
- sed ‘4q’ foo.txt
  * ”For each line, if that line is the fourth line, stop”
- sed -n ‘3,4p’ foo.txt
  * Since sed prints each line anyway, if we only want lines 3 & 4 (instead of all lines with lines 3 & 4 duplicated) we use the -n

sed: Line addressing (... continued)

- sed -n ‘$p’ foo.txt
  * ”For each line, if that line is the last line, print”
  * $ represent the last line
- Reversing line criteria (!)
- sed -n ‘3,$!p’ foo.txt
  * ”For each line, if that line is the third through last line, do not print it, else print”

sed: Context Addressing

- Use patterns/regular expressions rather than explicitly specifying line numbers
- sed -n ‘/^From: /p’ $HOME/mbox
  * retrieve all the sender lines from the mailbox file
  * ”For each line, if that line starts with ‘From’, print it.” Note that the / / mark the beginning and end of the pattern to match
- ls -l | sed -n ‘/^....w/p’
  * ”For each line, if the sixth character is a W, print”

sed: Substitution

- Strongest feature of sed
- Syntax is
  [address]s/expression1/string2/flag
- sed "$|l::/" data.txt
  * substitute the character ‘|’ with the character ‘:\’
- sed ‘s/\|:/\&’ data.txt
  * global
sed: Using files

- T Tedious to type in commands at the prompt, especially if commands are repetitive
- Can put commands in a file and sed can use them

```
sed -f cmds.sed data.txt
```

file with commands

awk

- Powerful pattern scanning and processing language
- Names after its creators Aho, Weinberger and Kernighan (Don't you love how commands are named?)
- Most commands operate on entire line
  - awk operates on fields within each line
- Usage:
  - awk options [scriptfile] file(s)
  - Example: awk -f awk.script foo.txt

awk: Processing model

BEGIN { command executed before any input is read}
{
    Main input loop for each line of input
}
END {commands executed after all input is read}

awk: First example

```sh
# Begin Processing
BEGIN {print "Print Totals"}

# Body Processing
{total = $1 + $2 + $3}
{print $1 " + " $2 " + " $3 " = "total}

# End Processing
END {print "End Totals"}
```

Input and output files

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 78 44</td>
<td>Print Totals</td>
</tr>
<tr>
<td>66 31 70</td>
<td>22 + 78 + 44 = 144</td>
</tr>
<tr>
<td>52 30 44</td>
<td>66 + 31 + 70 = 167</td>
</tr>
<tr>
<td>88 31 66</td>
<td>52 + 30 + 44 = 126</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>clothing 3141</td>
<td>computers 9161</td>
</tr>
<tr>
<td>computers 9161</td>
<td>textbooks 21312</td>
</tr>
<tr>
<td>textbooks 21312</td>
<td></td>
</tr>
</tbody>
</table>

awk 'if ($2 == "computers") {print}' sales.dat
awk: Other features

- Formatted printing using printf
- Conditional statements (if-else)
- Loops
  - for
  - while
  - do-while

awk: Associative arrays

- Normal arrays use integers for their indices
- Associative arrays with strings as their indices
- Example: Age["Robert"] = 56

awk: Example

```awk
# salesDeptLoop.awk script
BEGIN  {OFS = "\t"}
{deptSales [$2] += $3}
END {for (item in deptSales)
   {
      print item, ":", deptSales[item]
      totalSales += deptSales[item]
   } # for
   print "Total Sales", ":", totalSales
} # END
```

awk: Example

```awk
# salesDeptLoop.awk script
BEGIN  {OFS = "\t"}
{deptSales [$2] += $3}
END {for (item in deptSales)
   {
      print item, ":", deptSales[item]
      totalSales += deptSales[item]
   } # for
   print "Total Sales", ":", totalSales
} # END
```

Input and output

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  clothing 3141  computers : 21482</td>
<td>1  clothing 3141  computers : 21482</td>
</tr>
<tr>
<td>1  computers 9161  supplies : 2242</td>
<td>1  textbooks 21312  textbooks : 36774</td>
</tr>
<tr>
<td>2  clothing 3252  clothing : 6393</td>
<td>2  computers 12321  Total Sales : 66891</td>
</tr>
<tr>
<td>2  supplies 2242</td>
<td></td>
</tr>
<tr>
<td>2  textbooks 15462</td>
<td></td>
</tr>
</tbody>
</table>

Perl

- "Practical Extraction and Reporting Language"
- written by Larry Wall and first released in 1987
- rumour: name came first, then the acronym
- "Perl is a language for easily manipulating text, files and processes": originally aimed at systems administrators and developers
Features
- Enables quick development of programs
  - No need to define variable types
  - Portable
  - Extensible (module import/export mechanism)
  - Powerful "regular expression" capabilities
  - Simple I/O model
  - Many modules
  - Support for static scoping
  - Built-in debugger

Common uses
- Text-stream filters
  - Transforming, stripping, annotating, combining
  - Simple text manipulation
  - Common Gateway Interface (CGI) scripts
  - Report generation
  - System scripting
  - General solution prototyping
  - Hello, World!

Executing Perl scripts
- "Bang path" convention for scripts:
  - Can invoke Perl at the command line, or
  - Add `!/public/bin/perl` at the beginning of the script
  - Exact value of path depends upon your platform
    (use "which perl" to find the path)
- From the command line:

  ```perl
  % perl
  print "Hello, World!\n";
  CTRL-D
  Hello, World!
  ```

Basics
- Kinds of variable:
  - Scalars, lists, "hashes" (also called "associative arrays" or "dictionaries")
  - Some rudimentary support for object-orientation, but not really designed as an OOP language
  - Advanced Perl supports pointers, user-defined structures, subroutine references

Basics (cont'd)
- An example:

  ```perl
  #!/public/bin/perl
  $fruit{'apples'} = 5;
  $fruit{'oranges'} = 3;
  $fruit{'lemons'} = 2;
  $fruit{'limes'} = 2;
  @keys = keys($fruit);
  foreach $f (@keys) {
      print "We have $fruit{$f} $f\n";
  }
  ```

Control structures
- Similar to that in C:
  - If {} {} else {}
  - If {} {} elsif {} {} else {}
    (note spelling)
  - While ()
    do {} while()
    for (;;) {}
  - Foreach: iterates over each element in a list
  - No "switch" statement:
    - Must use sequence like "if-elsif-elsif-else"
  - Conditional expressions as in C:
    - Non-zero value: true
    - Zero value: false
using shell commands in Perl

- example:

```perl
$file_01 = "/home/foo/bar/ex1.txt";
$file_02 = "/home/foo/bar/ex2.txt";

$result = system ("diff $file_01 $file_02");
if (!$result == 0) {
    # files were the same
} else {
    # files were different
}
```

- If we are interested in only the result value and not the output from the command, redirect output to /dev/null
- example:

```perl
system("diff $file_01 $file_02 >/dev/null")
```