System Programming

sed, awk & perl

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 ‘1$ s/^ 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 ~~line~~s 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 's/|/:/' data.txt`
  - substitute the character '|' with the character ':'

substitute

- `sed 's/|/:/g' data.txt`

global

sed:Using files

- 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

```
# 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

- Input

```
22 78 44
66 31 70
52 30 44
88 31 66
```

Output

```
Print Totals
22 +78 +44 =144
66 +31 +70 =167
52 +30 +44 =126
88 +31 +66 =185
End Totals
```

awk:command line processing

- İnput

```
1 clothing     3141
1 computers    9161
1 textbooks   21312
2 clothing     3252
2 computers   12321
2 supplies     2242
2 textbooks   15462
```

- Output

```
1 computers 9161
2 computers 2321
```

```
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

```
# 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

- İnput

```
1 clothing     3141
1 computers    9161
1 textbooks   21312
2 clothing     3252
2 computers   12321
2 supplies     2242
2 textbooks   15462
```

- Output

```
Computers   :  9161
Supplies    :  2321
Textbooks   : 36774
Clothing    :  6393
Total sales: 66891
```

awk: Example

```
# 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
```

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!

```
print ("Hello,world!\n");
print "Hello,world!\n";
print STDOUT "Hello,world!\n";
```

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
 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 (contd)

- An example:

```
     #!/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 () {}`

    `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:

```
$file_01 =“/home/foobar/ex1.txt”;
$file_02 =“/home/foobar/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:...

```
system(“diff $file_01 $file_02 >/dev/null”)
```