CS 246 Spring 2018 — Tutorial 1

May 9, 2018

Summary

1 Shell Review

- Commands you should be familiar with:

  `cd` change the current directory
  - With no arguments or `~` returns you to your home directory
  - With `-` will return you to previous current directory

  `ls` list files in the current / specified directory
  - With `-l` returns long form listing of the files
  - With `-a` returns all (including hidden) files
  - With `-h` returns human readable format for various fields (e.g. file sizes: 100M, 1G)
  - Can combine multiple options, e.g. `ls -al`

  `pwd` prints the current directory
  - Same as `echo "$PWD"`

  `tail` print last 10 lines of file / standard input

`sort` sort lines of a file / standard input

- With `-n` option will sort strings of digits in numeric order

`uniq` removes consecutive duplicates (removes all duplicates if sorted)

- With `-c` option will print counts of consecutive duplicates

2 Output Redirection

- Suppose we have a program (`printer` — prints even numbers to stdout, odd to stderr) that prints to standard output and standard error.
- To redirect stdout to `print.out` and stderr to `print.err`:
  - `./printer > print.out 2> print.err`
- To redirect the output from standard output to standard error:
  - `echo "ERROR" >&2`
- To redirect standard output and standard error to the same file we need to tie them together. The following 3 are equivalent:
  - `./printer &> out`
  - `./printer > out 2>&1`
  - `./printer 2> out >&2`
- What would be the purpose of redirecting output to `/dev/null`?
  - When we do not care about the actual output of the program but want it to perform some operation (e.g. checking if files are the same, finished successfully).

3 Pipelining

- Suppose we want to determine the 10 most commonly occurring words in a collection of words (see `wordCollection` file) and output it to the file `top10`. How might we accomplish this?
  - Idea: use some combination of `sort` / `uniq` / `head` / `tail`. But how? Probably need `-c` option with `uniq` and `-n` option with `sort`.
  - Okay: `uniq -c wordCollection | sort -n`
  - But what's the problem? `wordCollection` isn't sorted!
  - So now we have: `sort wordCollection | uniq -c | sort -n`

  - So this gives us counts in least to most. How do we get the top 10 and output it to the file `top10`?
  - Let's add `tail` now: `sort wordCollection | uniq -c | sort -n | tail > top10`
- What if we wanted the word counts of the first 10 words alphabetically?
  - `sort wordCollection | uniq -c | sort -k2,2 | head > top10`
- What if we wanted the top 10 words but wanted to break ties based upon reverse alphabetical order?
  - `sort wordCollection | uniq -c | sort -k1,1nr -k2,2r | head > top10`

4 Embedded Commands

- We can use a subshell to use the output of commands as command line arguments to scripts.
- `egrep $(cat file) myfile.txt` could allow us to run `egrep` with the contents of a file being the regular expression.
- Note the difference between:
  - `echo "echo cat"` — the string `echo cat` is printed
  - `echo $(echo cat)` — the output of running the command `echo cat` is printed, which is the string `cat`

5 Types of Quotes

- Note that does not affect the way `egrep` evaluates regular expressions.

5.1 Double Quotes

- Suppresses globbing, but allows variable substitutions and embedded commands:
  - `echo *` — prints names of all files in the current directory
  - `echo "*"` — prints `*`
  - `echo "$(cat word.txt)"` — prints contents of `word.txt`
  - `echo "$HOME"` — prints the absolute path to the user's home directory

5.2 Single Quotes

- No substitution or expansion will take place with anything inside of single quotes.
- Suppresses globbing, variable substitution, and embedded commands:
  - `echo '*'` — prints `*`
  - `echo '$(cat word.txt)'` — prints `$(cat word.txt)`

Both single and double quotes can be used to pass multiple words as one argument. This is useful for e.g. passing file names with spaces in them.

6 `egrep` and Regular Expressions

- Recall that `egrep` allows us to find lines that match patterns in files / standard input.
- Some useful regular expression operators are:

`^`	matches the beginning of the line
`$`	matches the end of the line
`.`	matches any single character
`?`	the preceding item can be matched 0 or 1 times
`*`	the preceding item can be matched 0 or more times
`+`	the preceding item can be matched 1 or more times
`[...]`	matches any **one** of the characters in the set
`[^...]`	matches any one character not in the set
`\`	the character after this will be regarded as a character not an operator. i.e. `\.` matches the `.` character, instead of any single character.
`expr1|expr2`	matches `expr1` or `expr2`

- Recall that concatenation is implicit.
- Parentheses can be used to group expressions.
- The option `-n` will print line numbers.
- Give a regular expression to find lines starting with 'a' or ending with 'z':
  - `^a|z$`
- Give a regular expression to find lines with more than one occurrence of the characters a,e,i,o,u:
  - We may try `[aeiou](.*[aeiou])+`

  - But `[aeiou].*[aeiou]` would also suffice. Why?

- `egrep` can be especially useful for finding occurrences of variable / type names in source files. To find all lines containing the name `count` in all files ending in `.cc`:
  - `egrep "count" *.cc`
- **Remember:** regular expressions **are not the same as** globbing patterns.

7 Bash Example

- Create a Bash script called `mean` that is invoked as follows:

  `./mean filename`

  The argument `filename` is the name of a file containing a list of whitespace-separated numbers, from which the mean will be calculated.

8 Tips of the Week: Vim Basics

- You'll quickly notice that vim has a few basic modes. The one you are likely familiar with are the normal, insert, and command mode.
- If you get stuck and don't know what mode you are in, pressing `Esc` key a few times usually brings you back to normal mode.

8.1 Normal Mode

- In normal mode, most keys are hotkeys for various actions.
- For moving around:

  `C-f` (Ctrl + F) moves cursor one screen down.

  `C-b` (Ctrl + B) moves cursor one screen up.

  `w` moves cursor to the next word.

  `b` moves cursor to the previous word.

  `/` starts searching in the file. Enter the text to search and press `Enter` moves the cursor to the first match after cursor. To find the next match, press `n`.

- For editing text:

  `i` enters insert mode at the current position.

  `a` enters insert mode at the position after the current location.

`o` creates a new line after the current line, and enter insert mode.

`u` undoes last change.

8.2 Insert Mode

- This is the mode where you can write text. Anything you type will go into the file contents.
- Pressing `Esc` when you are in insert mode switches to normal mode.

8.3 Command Mode

- This is the mode that you enter by pressing `:` (colon) in normal mode.
- A colon will be shown on the bottom of the editor to indicate that you are in command mode.
- Similar to entering commands in a shell, you can use up / down arrow keys to go through the history, and press `Enter` to run a command.
- These are the most commonly used commands:

  `:q` closes vim if no changes have been made to the file.

  `:q!` closes vim without saving change which have been made to the file (since the last save).

  `:w` saves changes to the current file without quitting.

  `:wq` saves changes to the current file and closes vim.

  `:x` like `:wq`, but only save if changes have been made.