CS 246 Fall 2016 - Tutorial 1

September 23, 2016

1 Summary

- Shell Command Review
- I/O Redirection and Pipelining
- Regular Expressions and `egrep`
- Types of Quotes

2 Tip of the Week: Vi Shortcuts

You'll quickly notice that vi has a few basic modes. The one you are likely familiar with are the command and insert modes.

2.1 Insert Mode

- This is the mode where you can write text.
- Pressing escape when you are in command mode switchs to command mode

2.2 Command Mode

- During command mode, many keys are hotkeys for various action. The following list is a few useful hotkeys.
  - `i`: Enter insert mode where at the current position of the cursor.
  - `I`: Enter insert mode at the beginning of the line which the cursor is currently on.
  - `a`: Enter insert mode at the position after the current location of the cursor.
  - `A`: Enter insert mode at the end of the line which the cursor is currently on.
  - `o`: Enter insert mode on a new line after the line the cursor is currently on.
  - `O`: Enter insert mode on a new line before the line the cursor is currently on.
  - `x`: Delete the character the cursor is currently one.
  - `0`: Move the cursor to the beginning of the current line.
  - `$`: Move the cursor to the end of the current line.
  - `:q`: Close vi if no changes have been made to the file.
  - `:q!`: Close vi without saving change which have been made to the file.
  - `:wq`: Save changes to the current file and close vi.
  - `:n`: Move to line n of the file.
  - `:$`: Move to the end of the file.
  - `/<word>`: Searches for <word in the file and moves the cursor to the next occurrence of <word>. The <word> searched for can be a regular expression. Move to the next match by entering n. Move to the previous match by entering N.
  - `?<word>`: same as <word> with n and N's roles reversed.

3 Tip of the Week: Terminal

- Press on the up arrow to see previous commands
- The command `clear` will clear the terminal
- When typing in a command or file name, you can press tab to autocomplete the word if the remainder of the word is not ambiguous. Otherwise, it will fill in part of the word and pressing tab again will show the options for what word it could be.

4 `.snapshot`

- Sometimes we accidentally delete files that we wanted to keep
- In general, deleting a file with `rm` is unrecoverable
- However, on `linux.student.cs` your files are backed up for you
- The `.snapshot` directory in your home directory contains several sub-directories
- They correspond to hourly, nightly, and weekly backups of all your files on the undergrad environment
- You can navigate them to get an older versions of your files
- Note that you should copy (not move or edit) these files into your working directories
  - Otherwise, you risk deleting or otherwise modifying a backup

5 Shell Review

- Commands you should definitely know
- **cd** - change the current directory
  - With no directory or ~ returns you to your home directory
  - With - will return you to previous current directory
- **ls** - view files in the current/specified directory
  - With -l returns long form list of directory
  - 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 $PWD
- **uniq** - removes consecutive duplicates (removes all duplicates if sorted)
  - -c option will print counts of consecutive duplicates
- **sort** - sort lines of a file/standard in
  - -n option will sort strings of digits in numeric order
- **tail** - print last 10 lines of file/standard in

6 Output Redirection

- Suppose we have a program (printer - prints even numbers to stdout, odd to stderr) that prints to standard output and standard error. Give the redirection to redirect stdout to print.out and stderr to print.err.
  - `./printer > print.out 2> print.err`
- Suppose we want to redirect the output from standard output to standard error.
  - `echo "ERROR" 1>&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, executed correctly).

7 Pipelining

Suppose we want to determine the 10 most commonly occurring words in a collection of words (see wordCollection file) and output it to file top10. How might we accomplish this?

Idea: Use some combination of sort/uniq/tail. But how? Probably need -c option with uniq and sort -n.

Okay. `uniq -c wordCollection | sort -n`

But what's the problem? wordCollection isn't sorted!

So now: `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 try 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 -k2,2n | tail > top10
```

What if we wanted the top 10 words but wanted to break ties based upon reverse alphabetical order?

```
sort wordCollection | uniq -c | sort -k1,1n -k2,2r | tail > top10$
```

For fun (not covered material): wordCollection was created using the command:

```
# Randomly sample the system dictionary 10 times with varying sample sizes to create
# the word collection
for i in {1..10};
do
  sort -R /usr/share/dict/words | head -$((RANDOM % 10000)) >> wordCollection;
  # $((RANDOM % 10000)) access the global shell variable $RANDOM, which produces
  # a different (pseudo-)random number every time it is accessed, and reduces it
  # to the range of 0-10,000 using the modulus operator
done
```

8 `egrep` and Regular Expressions

- Recall that **egrep** allows us to find lines that match patterns in files
- 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
  - `[...]` - match one of the characters in the set
  - `[^..]` - match one character not in the set
  - `expr1|expr2` - match expr1 or expr2

- Recall that concatenation is implicit
- Parentheses can be used to group expressions
- egrep can be especially useful for finding occurrences of variable/type names in source files
  - The option -n will print line numbers
- The following are some examples:
  - `egrep "count" file.c`
  - `egrep "count" *.c`
  - An obvious thign to do is find where in these file count occurs: `egrep -n "count" *.c`
  - Give a regular expression to find all lines starting with 'a' and ending with 'z'.
    - `egrep "^a.*z$" /usr/share/dict/words`
  - Give a regular expression to find lines starting with 'a' or lines ending with 'z'.
    - `egrep "^a|z$" /usr/share/dict/words`
  - Give a regular expression to find lines with one or more occurrences of the characters a,e,i,o,u
    - `egrep "[aeiou]" /usr/share/dict/words`
  - Give a regular expression to find lines with more than one occurrence of the characters a,e,i,o,u
    - `egrep "[aeiou].*[aeiou]+" /usr/share/dict/words`

9 Types of Quotes

9.1 Double Quotes

- Suppresses globbing
- Allows variable and command substitution

```
echo *   # returns names of all files in the current directory
echo "*" # returns *
echo "$(cat word.txt)" # will print contents of word.txt
echo "${HOME}" # will print the absolute path to the user's home directory
```

9.2 Single Quotes

- No substitution or expansion will take place with anything inside of single quotes.
- Suppresses variables and comamnd substitution

```
echo '$(cat word.txt)' #Will print $(cat word.txt)
```

Both single and double quotes can be used to pass multiple words as one argument. This is required for opening files and directories with spaces in their names.