CS 246 Fall 2013 - Tutorial 1

September 17, 2013

1 Summary

- General Administration Stuff
- Terminal Setup
- Text Editors
- Shell Command Review
- I/O Redirection and Pipelining
- Regular Expressions and `grep`

2 General Administration Stuff

- Course E-mail: `cs246@linux.student.cs`
- Rob's Office Hours: MC 4065 - Hours M 10:30 - 12:00, F 11:00 - 12:00
- Use `Piazza` for most questions
  - Questions with potential answers should be private or asked in office hours
  - If your question is made private by an instructor - keep it that way
- E-mail the course account or post on Piazza about topics you would like to see in upcoming tutorials

3 Terminal Setup

3.1 XServer

- To run X (read: graphical) applications, your computer needs to be running an X Server
- Linux installations by default are running one
- OSX installations may or may not be (depends on OSX version)
  - See the XQuartz project for more information.
  - `http://xquartz.macosforge.org/`
- Windows will also require an XServer - we recommend XMing
  - See `http://sourceforge.net/projects/xming/`

3.2 Terminal - OSX, Cygwin, Linux

- Open the terminal
- Enter the command: `ssh -Y your-username@linux.student.cs.uwaterloo.ca`
  - Note that the -Y option allows for X11 forwarding (e.g. graphical applications)
  - Username refers to your Quest username
- Enter your Quest password
  - Should this not work. You need to reset your password.
  - Visit `http://www.student.cs.uwaterloo.ca/password`
- Note that your default shell may not be bash. To make your default shell bash. Visit the password reset site as well.

3.3 PuTTY

- Open PuTTY
- In the **Host Name** field enter `linux.student.cs.uwaterloo.ca`
- In the sidebar under **SSH**, click **X11**
- Click the box that says `Enable X11 forwarding`
- Press `Open`
- Enter your Quest username and password
  - It may appear that nothing is happening when you type your password but things are

4 Text Editors

- There are several different command line text editors that you can use.
  - Vi/Vim is what we recommend and will give you less grief during the course
  - Other options are Emacs, pico, nano. Emacs has issues with whitespace and pico/nano are very simple.
  - Countless debates have arisen between Vi and Emacs. We're not going to get into the trade-offs
- A quick Vim rundown:
  - Enter the command `vim file` to create or edit an existing file named `file`
  - By default Vim starts in command mode
  - Different keystrokes activate different commands
    - **h,j,k,l** - navigate like the arrow keys (which generally work in Vim as well)
    - **x** - delete highlighted character
    - **r** - replace highlighted character with next key pressed
    - **o** - create a line below the current line
    - **O** - create a line above the current line
    - **i** - start insert mode (can enter text - arrow keys may or may not work)
    - **Esc** - return to normal mode
    - **:w file** - save the file to given filename
      - If no filename given, then write to filename specified when Vim was opened
    - **:wq** - write and quit
    - **:q** - quit
  - Use `vimtutor` to learn all you need to know about Vim
    - Enter `vimtutor` on the command line to enter the tutorial

5 Shell Review

- Commands you should definitely know
- **cd** - change the current directory
  - With no directory or ~returns you to your home directory ($HOME)
  - 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
  - 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 and Piping

6.1 Basic Examples

- How do we redirect standard output to standard error?
  - `1>&2`
- Suppose we have a program (printer) 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`
- 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).
- What is the difference between `./printer 2>&1 > out` and `./printer > out 2>&1`
  - The first prints all odd numbers to stdout and even numbers to the file out
  - The second prints all numbers to the file out

6.2 More Complex Example

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`

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

```
for i in ‘seq 1 10‘;
do
num=$((RANDOM % 10000));
echo $num;
sort -R /usr/share/dict/words | head -n $num >> wordCollection;
done
```

7 grep and Regular Expression

- Recall that **grep** allows us to find lines that match patterns in files
- To get the full power of regular expressions supported, we must use **egrep** or **grep -E**
- Some useful regular expression operators are:
  - `^` - the pattern following must be at the beginning of the line
  - `$` - the pattern preceeding must be at 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
  - `+` - one or more times
  - `[...]` - match one of the characters in the set
  - `[^..]` - match a character not the set
  - `expr1|expr2` - match expr1 or expr2
- Also, recall that concatenation is implicit
- Note that parentheses can be used to group expressions
- egrep can be especially useful for finding occurrences of names in source files
  - The option -n will print line numbers
- The following are some examples:

```
> egrep "count" file.c
> egrep "count" *.c
> 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
> # Want all lines in all .c files that modify count by assigning either 0 or 1 aside from initialization
> # Let’s try the obvious thing first
> egrep "^ *count *= *0|1;$" *.c
> # Doesn’t work. Why?
> # Let’s use parenthesis
> egrep "^ *count *= *(0|1) *; *$" *.c
> # Excellent, this works.
```