CS 246 Fall 2015 - Tutorial 1

September 25, 2015

1 Summary

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

2 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

3 Output Redirection and Piping

3.1 Basic Examples

- 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`
- What if we want to redirect standard output and standard error to the same file?

  - Will `./printer > out 2> out` work?

- To print to standard out and standard error to the same file we need to tie them together.
  - For example, `./printer &> out`, which prints both stdout and stderr to out
  - Or `./printer > out 2>&1`
  - Or `./printer 2> out 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).

- 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
  - Order of redirection matters!

3.2 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 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
```

4 grep and Regular Expressions

- Recall that **grep** allows us to find lines that match patterns in files
- To get the full power of regular expressions, we must use **egrep** or **grep -E**
- 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
> 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.
```

5 Types of Quotes

5.1 Double Quotes

- Suppresses globbing

```
echo *   # returns names of all files in the current directory
echo "*" # returns *
```

5.2 Back Quotes

- executes the command in the quotes

```
egrep `cat word.txt` /usr/share/dict/words
```

- This command will return every word in the dictionary which matches the word in word.txt (Note: This is a partial match, 'the' would match 'other').

5.3 Single Quotes

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

```
echo ’‘cat word.txt‘’
> ‘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.