Text Manipulation in UNIX

Reading

One of the most common commands in UNIX for reading a text file is the 'cat' command, which is short for conCATenate. It takes all the text content from a file and concantenates them as readable text on the screen of the terminal.

```
cat poem.txt
```

You can add line numbers by adding the the -n option

```
cat -n poem.txt
```

User Input

There is another another command for accepting user imput called 'read', which we will look at when we study bc. This should not be confused with 'cat'.

Writing

To write contents to the screen you can use the 'echo' command.

```
echo This is a line I typed
```

You see this simply repeats or echos what you wrote. It doesn't appear to be useful at first, but with other operators in UNIX it can become useful.

```
echo This is a line I typed and now it is stored > storedLine.txt
```

Now if you list your current directory you will see you have a new file called storedLine.txt

```
ls
```

Read it

```
cat storedLine.txt
```

The '>' is a i/o or redirection operator in UNIX. You can also use '>>' following any command to output the command to somewhere else. Another redirection operator in the '|' or pipe.

```
echo This is a line I spoke | say
```

Here is another example of pipelining

```
cat ballad.txt | more
```

Now keep hitting enter and you will scroll forward line-by-line or spacebar to scroll screen-by-screen

```
cat ballad.txt | less
```

'less' is more flexible, it allows you to us the arrows or 'B' and 'Q'

Sorting

UNIX can arrange items in alphanumerical order with the 'sort' command including thos in text files

```
cat names.txt
```

Then...

```
sort names.txt
```

Lines of text from multiple files can be merged and sorted into one display

```
sort names.txt fruit.txt
```

If you want to display AND save to a new file, instead of > or >> you can pipe it through to the 'tee' command

```
sort names.txt fruit.txt | tee sorted.txt
```

Search, Replace and in-Line Editing

UNIX has two very powerful and useful utilities, 'grep' and 'sed'. Especially when these are paired with Regular Expressions that we studied earlier. The grep tool is used for searching with regex patterns to find files that match. So, 'grep' copies to output those lines in the input that matches a specified pattern. We will not go into too much detail about grep in this class, but here is a useful trick that employs 'grep', 'more' and '|' that I often use.

I happen to be a very bad speller. So bad, in fact, that I often do not even know how to spell some words close enough for spell checks to even recognize what I WANT to spell. I have a trick, however. In UNIX and Linux distros there is a dictionary file. This can useful for testing passwords and other things. Now, my spelling problem is Ludicrous(sp?) I know, but I do know that it starts with lud. Let me see how it is spelled. My dictionary example is for Mac OSX UNIX.

```
more /usr/share/dict/words | grep ^lud | more
```

Phew! I spelled it correctly. Now I didn't need that final pipe through more at the end for this particular search, but I was not sure how many words in the dict file started with lud and I didn't want too many words on my terminal screen.

The other command is 'sed' or Stream EDitor. It allows for editing text files from the command lind or "in-line." The most common use of 'sed' is to replace words with other words. Usually this is done with text from a text file, but we'll do a couple of examples first using 'echo'

```
echo good day | sed 's/day/night/'
```

The 's' before /day is the substitution command. We can also use the 's' command to add puctuation. Let's add a set of parantheses around a word. The first occurance of any lowercase string on each line can be matched using '[a-z]*' As the exact string MAY be unknown in this case the editing command allows an '&' to represent the string in the replacement string section (after the second '/'). All lowercase strings can be matched on each line by appending a 'g' at the end of the command, this stands for Global.

```
echo hot day | sed 's/[a-z]*/(&)/g'
```

Then...

```
echo hot day | sed 's/[a-z]*/(&)(&)/g'
```

And finally to put a set around both words.

```
echo hot day | sed 's/[a-z]*/(&)/g'
```

Now, a more practical use of 'sed' is to edit text files. We do this by following the sed command with the '-e' option and putting the input file after the pattern.

```
sed -e 'pattern' file.txt
```

Take the bio of Benjamin Franklin in your folder and make you own. I'll use my name but you use yours in your pattern. Let's look at the file first.

```
cat FranklinBio.txt
```

Now let us replace Ben's name with ours. Now we want to replace both the first and last name. We can do this by using two option '-e' and patterns

```
sed -e 's/Benjamin/Stephen/g' -e 's/Franklin/Sabaugh/g' FranklinBio.txt
```

Note, this will only edit the stream to the display. The original file remains unchanged.

```
cat FranklinBio.txt
```

To save our plagiarized bio we can use the methods we learned earlier. Here is one option...

```
sed -e 's/Benjamin/Stephen/g' -e 's/Franklin/Sabaugh/g' FranklinBio.txt >> SabaughBio.txt
```