Session 5

We’ll now look at how to combine programs.

We’ll start with the directory Data\_Files\_TMCS/molecules.

This contains six files describing some simple organic molecules. The .pdb extension indicates that these files are in Protein Data Bank format, a simple text format that specifies the type and position of each atom in the molecule.

$ ls molecules

cubane.pdb ethane.pdb methane.pdb

octane.pdb pentane.pdb propane.pdb

Try running the ‘word count’ command wc cubane.pdb. This command counts the number of lines, words, and characters in files (from left to right, in that order).

$ cd molecules

$ wc cubane.pdb

20 156 1158 cubane.pdb

By combining it with a wild card, wc \*.pdb, the \* in \*.pdb matches zero or more characters, so the shell turns \*.pdb into a list of all .pdb files in the current directory:

$ wc **\***.pdb

20 156 1158 cubane.pdb

12 84 622 ethane.pdb

9 57 422 methane.pdb

30 246 1828 octane.pdb

21 165 1226 pentane.pdb

15 111 825 propane.pdb

107 819 6081 total

By running wc -l instead of just wc, the output shows only the number of lines per file:

$ wc **-l** **\***.pdb

20 cubane.pdb

12 ethane.pdb

9 methane.pdb

30 octane.pdb

21 pentane.pdb

15 propane.pdb

107 total

The -m and -w options can also be used with the wc command, to return the number of characters or the number of words in the files respectively.

If wc is called and no filename is given it will sit there and wait. To get out of this, and any other issues you may face, press ctrl+c. This will cancel any process that is currently happening.

Lets say we wanted to see which file had the fewest lines. If we had thousands of files it wouldn’t be so easy as it is now. To find the answer we start by using the > symbol to redirect a commands output to a file instead of printing it to screen.

$ wc **-l** **\***.pdb > lengths.txt

We can now send the content of lengths.txt to the screen using cat lengths.txt. The cat command gets its name from ‘concatenate’ i.e. join together, and it prints the contents of files one after another. There’s only one file in this case, so cat just shows us what it contains:

$ cat lengths.txt

20 cubane.pdb

12 ethane.pdb

9 methane.pdb

30 octane.pdb

21 pentane.pdb

15 propane.pdb

107 total

Cat is a very useful command however it always prints the whole file to screen. In practise less is sometimes more useful. If you used it in place of Cat you will only see one screens amount of the file, instead of the whole thing.

We can now use sort – n to organise the file. The n specifies a numerical sort and not an alphabetic one. This also sends the results to screen.

$ sort **-n** lengths.txt

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

107 total

We can put the sorted list of lines in another temporary file called sorted-lengths.txt by putting > sorted-lengths.txt after the command, just as we used > lengths.txt to put the output of wc into lengths.txt. Once we’ve done that, we can run another command called head to get the first few lines in sorted-lengths.txt:

$ sort **-n** lengths.txt > sorted-lengths.txt

$ head **-n** 1 sorted-lengths.txt

9 methane.pdb

Using -n 1 with head tells it that we only want the first line of the file; -n 20 would get the first 20, and so on. Since sorted-lengths.txt contains the lengths of our files ordered from least to greatest, the output of head must be the file with the fewest lines.

### Quick Questions:

1. If you were to run code with a single > multiple times the file would be overwritten each time. To add new information to the file use the double >>. Try this yourself by running the below commands multiple times and studying the file that is produced?
2. What did the echo command do here?

$ echo The echo command prints text

$ echo hello > testfile01.txt

$ echo hello >> testfile02.txt

### Appending Data

### Quick Questions:

### What would the below code return?

$ head **-n** 3 animals.txt > animals-subset.txt

$ tail **-n** 2 animals.txt >> animals-subset.txt

If you found the above question hard to get your head round we can do some things to make it clearer. Below we run sort and head together

$ sort **-n** lengths.txt | head **-n** 1

9 methane.pdb

The vertical bar, |, between the two commands is called a pipe. It tells the shell that we want to use the output of the command on the left as the input to the command on the right.

Nothing prevents us from chaining pipes consecutively. That is, we can for example send the output of wc directly to sort, and then the resulting output to head. Thus we first use a pipe to send the output of wc to sort:

$ wc **-l** **\***.pdb | sort **-n**

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

107 total

And now we send the output of this pipe, through another pipe, to head, so that the full pipeline becomes:

$ wc **-l** **\***.pdb | sort **-n** | head **-n** 1

9 methane.pdb

### Quick Questions:

1. In our current directory, we want to find the 3 files which have the least number of lines. Which command listed below would work?

wc -l \* > sort -n > head -n 3

wc -l \* | sort -n | head -n 1-3

wc -l \* | head -n 3 | sort -n

wc -l \* | sort -n | head -n 3

Unix’s ability to link lots of simple tools together is what makes it so useful. This programming model is called ‘pipes and filters’. We’ve covered pipes and now we will look at filters. A filter is a program like wc or sort that transforms a stream of input into a stream of output. Just like most all other Unix programs, it reads from standard input, performs an operation or a series of operations, and writes to standard output.

The key is that any program that reads lines of text from standard input and writes lines of text to standard output can be combined with every other program that behaves this way as well. You can *and should* write your programs this way so that you and other people can put those programs into pipes to multiply their power.

### Quick Questions:

1. A file called animals.txt (in the Data\_Files\_TMCS /data folder) contains the following data:

2012-11-05,deer

2012-11-05,rabbit

2012-11-05,raccoon

2012-11-06,rabbit

2012-11-06,deer

2012-11-06,fox

2012-11-07,rabbit

2012-11-07,bear

What text passes through each of the pipes and the final redirect in the pipeline below?

$ cat animals.txt | head **-n** 5 | tail **-n** 3 | sort **-r** > final.txt

Hint: build the pipeline up one command at a time to test your understanding

1. Pipe Construction

For the file animals.txt from the previous exercise, consider the following command:

$ cut **-d** , **-f** 2 animals.txt

The cut command is used to remove or ‘cut out’ certain sections of each line in the file, and cut expects the lines to be separated into columns by a Tab character. A character used in this way is a called a **delimiter**. In the example above we use the -d option to specify the comma as our delimiter character. We have also used the -f option to specify that we want to extract the second field (column). This gives the following output:

deer

rabbit

raccoon

rabbit

deer

fox

rabbit

bear

The uniq command filters out adjacent matching lines in a file. How could you extend this pipeline (using uniq and another command) to find out what animals the file contains (without any duplicates in their names)?

1. Which Pipe?

The file animals.txt contains 8 lines of data formatted as follows:

2012-11-05,deer

2012-11-05,rabbit

2012-11-05,raccoon

2012-11-06,rabbit

...

The uniq command has a -c option which gives a count of the number of times a line occurs in its input. Assuming your current directory is Data\_Files\_TMCS/data/, what command would you use to produce a table that shows the total count of each type of animal in the file?

1. sort animals.txt | uniq -c
2. sort -t, -k2,2 animals.txt | uniq -c
3. cut -d, -f 2 animals.txt | uniq -c
4. cut -d, -f 2 animals.txt | sort | uniq -c
5. cut -d, -f 2 animals.txt | sort | uniq -c | wc -l
6. Wildcard Expressions

Use the directory Data\_Files\_TMCS /north-pacific-gyre/2012-07-03. The wildcard \*[AB].txt matches all files ending in A.txt or B.txt. Imagine you forgot about this.

1. Can you match the same set of files with basic wildcard expressions that do not use the [] syntax? *Hint*: You may need more than one command, or two arguments to the ls command.
2. If you used two commands, the files in your output will match the same set of files in this example. What is the small difference between the outputs?
3. If you used two commands, under what circumstances would your new expression produce an error message where the original one would not?
4. Removing Unneeded Files

Suppose you want to delete your processed data files, and only keep your raw files and processing script to save storage. The raw files end in .dat and the processed files end in .txt. Which of the following would remove all the processed data files, and *only* the processed data files?

1. rm ?.txt
2. rm \*.txt
3. rm \* .txt
4. rm \*.\*

**Takeaways**

* cat displays the contents of its inputs.
* head displays the first 10 lines of its input.
* tail displays the last 10 lines of its input.
* sort sorts its inputs.
* wc counts lines, words, and characters in its inputs.
* command > file redirects a command’s output to a file (overwriting any existing content).
* command >> file appends a command’s output to a file.
* first | second is a pipeline: the output of the first command is used as the input to the second.
* The best way to use the shell is to use pipes to combine simple single-purpose programs (filters).