Session 6

### Loops

### A loop causes the code to repeat a command, or a string of commands, many times. They allow you to operate on huge numbers of files automatically. Enter the creatures directory and look at the files.

head **-n** 5 basilisk.dat minotaur.dat unicorn.dat

The classification for each species is given on the 2nd line of the files. To print this to screen we would have to take two steps: head -n 2 and pipe it to tail -n 1. A loop can mean this is carried out automatically.

**for** item **in** set\_of\_items

**do**

some\_process\_on $something *# Unlike Python, indentation isn’t*

**done**

A more direct example:

$ **for** filename **in** basilisk.dat minotaur.dat unicorn.dat

>

> **do**

> head **-n** 2 $filename | tail **-n** 1

> **done**

CLASSIFICATION: basiliscus vulgaris

CLASSIFICATION: bos hominus

CLASSIFICATION: equus monoceros

For carries out a similar role in bash to what it does in Python. It causes the program to iterate over a list executing commands. The $ informs the shell to treat the variable as a name and change it as needed.

In the first iteration the head command is run on the first file, and the first two lines are piped to the tail command which then prints the output to screen. The second iteration moves to the next filename. When there are no files left the programme ends.

As you can see spaces are used to separate elements of a list when looping. If an element contains a space character, for instance ‘basilisk size.dat’ then we would need to surround it with double quotes. Its easier to just avoid putting spaces in names, use underscores instead.

### Explaining the prompt

The keyword for tells the shell to repeat a command (or a set of commands) once for every item in a list. Every iteration, an item in the list is assigned to the variable, and the loops commands are executed. The next item in the list is then operated on. A variables value is called by putting a $ in front of it in the loop.

Each time the loop iterates, a file name will be added to the variable filename and run the head command. On the 1st loop,  basilisk.dat is $filename. The head command  is run on basilisk.dat and the first two lines are piped to the tail command, which then prints the second line of basilisk.dat. In the 2nd iteration, $filename becomes minotaur.dat and the shell runs head on minotaur.dat before piping the first two lines to the tail command, which prints the second line of minotaur.dat. In the 3rd iteration, $filename is unicorn.dat, so the shell runs the head command on that file, and tail on the output of that. The loop then ends.

### Quick Questions:

### Go to the Data\_Files\_TMCS/molecules directory.

1. What is the output of the following code?

$ **for** datafile **in** **\***.pdb

> **do**

> ls **\***.pdb

> **done**

1. What is the output of the following code?

$ **for** datafile **in** **\***.pdb

> **do**

> ls $datafile

> **done**

1. What is the reason for the 2 different outputs?
2. Limiting Sets of Files
3. What would be the output of running the following loop in the Data\_Files\_TMCS/molecules directory?

$ **for** filename **in** c**\***

> **do**

> ls $filename

> **done**

1. How would the output differ from using this command instead?

$ **for** filename **in** **\***c**\***

> **do**

> ls $filename

> **done**

1. In the Data\_Files\_TMCS/molecules directory, what effect does this loop have?

**$ for** alkanes **in** **\***.pdb

**>do**

**>** echo $alkanes

> cat $alkanes > alkanes.pdb

**>done**

1. In the same directory as Q4, what would be the output of the following loop?

**>for** datafile **in** **\***.pdb

**>do**

**>** cat $datafile >> all.pdb

**>done**

1. Modify each of the files in Data\_Files\_TMCS /creatures, and then save a version of the original files, whilst naming the copies original-basilisk.dat and original-unicorn.dat. Use something like the following

$ **for** filename **in** **\***.dat

> **do**

> cp $filename original-$filename

> **done**

1. Explicitly write out the steps that this programme is taking

**History**

We can use the history command to get a list of the last few hundred commands that have been executed, and then to use !x (x = the command number) to repeat a commands.

There are a number of other shortcut commands for getting at the history.

* Ctrl+R enters a history search mode ‘reverse-i-search’ and finds the most recent command in your history that matches the text you enter next. Press Ctrl+R one or more additional times to search for earlier matches. You can then use the left and right arrow keys to choose that line and edit it then hit Return to run the command.
* !! retrieves the immediately preceding command (you may or may not find this more convenient than using ↑)
* !$ retrieves the last word of the last command. That’s useful more often than you might expect: after bash goostats NENE01729B.txt stats-NENE01729B.txt, you can type less !$ to look at the file stats-NENE01729B.txt, which is quicker than doing ↑ and editing the command-line.

### Quick Questions:

1. One way to check if a loop is doing what we’ve told it is to echo the commands it would run instead of actually running them.

What is the difference between the two loops below, and which one would we want to run?

*# Version 1*

$ **for** datafile **in** **\***.pdb

> **do**

> echo cat $datafile >> all.pdb

> **done**

*# Version 2*

$ **for** datafile **in** **\***.pdb

> **do**

> echo "cat $datafile >> all.pdb"

> **done**

1. Suppose we want to set up a directory structure to organize some experiments measuring reaction rate constants with different compounds *and* different temperatures. What would be the result of the following nested loops:

$ **for** species **in** cubane ethane methane

> **do**

> **for** temperature **in** 25 30 37 40

> **do**

> mkdir $species-$temperature

> **done**

> **done**

**Takeaways**

* A for loop repeats commands once for every thing in a list.
* Every for loop needs a variable to refer to the thing it is currently operating on.
* Use $name to expand a variable (i.e., get its value). ${name} can also be used.
* Do not use spaces, quotes, or wildcard characters such as ‘\*’ or ‘?’ in filenames, as it complicates variable expansion.
* Give files consistent names that are easy to match with wildcard patterns to make it easy to select them for looping.
* Use the up-arrow key to scroll up through previous commands to edit and repeat them.
* Use Ctrl+R to search through the previously entered commands.
* Use history to display recent commands, and !number to repeat a command by number.