Session 7

In this session we will look into how we might group useful commands together in a file and save them as a script. This script can then carry out operations and is termed a program. Therefore, the aim of this session is to use all the knowledge we have gained from the previous sessions and build a program!

Change the working directory to molecules/. Once this has been done, use the nano text editor to create a new file called program\_1.sh. Using the nano text editor write code that will select lines 3-10 of octane.pdb. If you are struggling with this, refer back to previous sessions, or check below for the answer.

By writing it in the file we are not executing any commands. If we save this file (Ctrl-O in nano) and exit (Ctrl-X in nano), we should then have created a program called program\_1.sh that we can call.

We can then execute the file in the terminal with the following command:

$ bash program\_1.sh

The output from the program is exactly what we would have got if we had ran the pipeline directly.

Congratulations, you’ve just created and ran a program!

**Answer:**

head -n 10 octane.pdb | tail -n 8

We can make this much cleverer though. Say we needed to select lines from an arbitrary file. We don’t want change the filename inside program\_1.sh each time as this is laborious. We should instead edit the program to automate this task.

$ nano program\_1.sh

Now, within “nano”, replace the text octane.pdb with the special variable called $1.

head -n 10 "$1" | tail -n 8

$1 is really clever. It stands for ‘the first filename (or other argument) on the command line’. We can now run our script like this:

$ bash program\_1.sh octane.pdb

or on a different file like this:

$ bash middle.sh pentane.pdb

We now have a much more versatile program!

**Double-Quotes Around Arguments**

For the same reason that we put the loop variable inside double-quotes, in case the filename happens to contain any spaces, we surround $1 with double-quotes.

We still need to edit program\_1.sh each time we want to adjust the range of lines, though. Let’s fix that by using the special variables $2 and $3 for the number of lines to be passed to head and tail respectively:

$ nano program\_1.sh

head -n "$2" "$1" | tail -n "$3"

We can now run:

$ bash program\_1.sh pentane.pdb 15 5

ATOM 9 H 1 1.324 0.350 -1.332 1.00 0.00

ATOM 10 H 1 1.271 1.378 0.122 1.00 0.00

ATOM 11 H 1 -0.074 -0.384 1.288 1.00 0.00

ATOM 12 H 1 -0.048 -1.362 -0.205 1.00 0.00

ATOM 13 H 1 -1.183 0.500 -1.412 1.00 0.00

By changing the arguments to our command we can change our script’s behaviour:

$ bash program\_1.sh pentane.pdb 20 5

ATOM 14 H 1 -1.259 1.420 0.112 1.00 0.00

ATOM 15 H 1 -2.608 -0.407 1.130 1.00 0.00

ATOM 16 H 1 -2.540 -1.303 -0.404 1.00 0.00

ATOM 17 H 1 -3.393 0.254 -0.321 1.00 0.00

TER 18 1

This works, but it may take the next person who reads program\_1.sh a moment to figure out what it does. We can improve our script by adding some comments at the top (just like in Python). A comment starts with a # (again just like in Python).

$ nano program\_1.sh

# Select lines from the middle of a file.

# Usage: bash program\_1.sh filename end\_line num\_lines

head -n "$2" "$1" | tail -n "$3"

What if we want to process many files in a single pipeline? For example, if we want to sort our .pdb files by length, we would type:

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

because wc -l lists the number of lines in the files (recall that wc stands for ‘word count’, adding the -l option means ‘count lines’ instead) and sort -n sorts things numerically. We could put this in a file, but then it would only ever sort a list of .pdb files in the current directory.

If we want to be able to get a sorted list of other kinds of files, we need a way to get all those names into the script. We can’t use $1, $2, and so on because we don’t know how many files there are. Instead, we use the special variable $@, which means, ‘All of the command-line arguments to the shell script’. We also should put $@ inside double-quotes to handle the case of arguments containing spaces ("$@" is special syntax and is equivalent to "$1" "$2" …).

Here’s an example:

$ nano sorted.sh

# Sort files by their length.

# Usage: bash sorted.sh one\_or\_more\_filenames

wc -l "$@" | sort -n

$ bash sorted.sh **\***.pdb ../creatures/**\***.dat

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

163 ../creatures/basilisk.dat

163 ../creatures/minotaur.dat

163 ../creatures/unicorn.dat

596 total

### Quick Questions:

1. Suppose you have several hundred data files, each of which is formatted like this:

2013-11-05,deer,5

2013-11-05,rabbit,22

2013-11-05,raccoon,7

2013-11-06,rabbit,19

2013-11-06,deer,2

2013-11-06,fox,1

2013-11-07,rabbit,18

2013-11-07,bear,1

We can use the command cut -d , -f 2 animals.txt | sort | uniq to produce the unique species in animals.txt. In order to avoid having to type out this series of commands every time, a scientist may choose to write a shell script instead.

Write a shell script called species.sh that takes any number of filenames as command-line arguments, and uses a variation of the above command to print a list of the unique species appearing in each of those files separately.

1. In the molecules directory, imagine you have a shell script called script.sh containing the following commands:

head **-n** $2 $1

tail **-n** $3 $1

While you are in the molecules directory, you type the following command:

bash script.sh '\*.pdb' 1 1

Which of the following outputs would you expect to see?

1. All of the lines between the first and the last lines of each file ending in .pdb in the molecules directory
2. The first and the last line of each file ending in .pdb in the molecules directory
3. The first and the last line of each file in the molecules directory
4. An error because of the quotes around \*.pdb
5. Write a shell script that takes a directory name along with a filename extension as its arguments before printing out the name of the file with the least lines in that directory with that extension.
6. Navigate back to the Data\_Files\_TMCS /molecules directory. Look at the three scripts below. What would they do when run as bash script1.sh \*.pdb, bash script2.sh \*.pdb, and bash script3.sh \*.pdb .

*# Script 1*

echo **\***.**\***

*# Script 2*

**for** filename **in** $1 $2 $3

**do**

cat $filename

**done**

*# Script 3*

echo $@.pdb

**Takeaways**

* Save commands in files (usually called shell scripts) for re-use.
* bash filename runs the commands saved in a file.
* $@ refers to all of a shell script’s command-line arguments.
* $1, $2, etc., refer to the first command-line argument, the second command-line argument, etc.
* Place variables in quotes if the values might have spaces in them.
* Letting users decide what files to process is more flexible and more consistent with built-in Unix commands.