Introduction to shell and python scripting

Contents

1 Introduction

Almost all serious scientific computing occurs within a unix operating system environment. There is a significant 'activation energy barrier' to start using unix, because it uses a text-based interface rather than a graphical user interface *e.g.* transferring files requires typing in a command, rather than opening up a window or two, and dragging and dropping using a mouse.

However, this apparent weakness is also unix's main strength. In particular, it becomes easy to automate repetitive tasks, to break the 'user input' bottleneck *i.e.* to make sure it is the computer working hard, rather than you! Ideally, your computer should be 100% occupied running calculations all the time, leaving you free to harvest data, process it, write papers, do multiple jobs/projects in parallel - just generally be more efficient all around.

Exercise: think of all the things you do repeatedly in your work *e.g.* setting up input files, running calculations, *etc.* Which of these would be most helpful to automate and why? Which would be easiest and hardest to automate? Write down the steps you now do by hand but would like to have the computer do for you. Share your thoughts with the group.

2 The power of unix

If you are unfamiliar with unix, the best place to start is opening up a unix terminal (or Cygwin window, if you've got a PC that doesn't have a linux partition), and work through the tutorial here (up to and including page 6):

www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html

This will basically ensure that you are able to see what's in folders, get around between them, make and remove folders and files, move or copy files or folders, view the contents of and search for text in files, append the contents of one file to another, work out what programs you have running. It may be helpful to tabulate these commands in your notebook for future reference, particularly if you are not particularly familiar with using unix. In addition, a UNIX 'cheat-sheet' has been provided separately, detailing simple and more advanced command-line tools.

A few additional helpful commands:

echo "blah blah blah" >> file	adds line "blah blah blah" to file
which program	tells you where 'program' lives in your file system
perl -pi -e 's/xxx/yyy/g' file	replaces all instances of 'xxx' with 'yyy' in file

3 Making and editing files the vi text editor

Like unix itself, there is a bit of an 'activation energy barrier' to using an exclusively text-based editing program like vi. However, again, the effort is worth the payoff, particularly if you want to really boost your research productivity. For honours students, the time investment may not be worth it, but for PhD students, definitely!

To start using the vi editor, it's pretty straightforward, just type:

vi filename

If filename refers to an existing file, you will see the contents of that file come up on your screen. Otherwise, vi will create a new, empty file with the name specified. Once you are in the file, you need to be able to do two things;

1. enter text,
2. issue commands (copy, paste, insert, delete *etc.*)

The vi editor opens in 'command mode'. To start typing in a new document, press 'i' (for insert) and you're on your way.

If you're editing an existing document, simply scroll down with the arrow keys (or type the number of lines you want to go down, then press the down button), and either press:

- 'i' to insert new text
- 'x' to delete a single character
- a number and then 'x' to delete a number of characters
- 'r' to overwrite a single character
- 'R' to overwrite multiple characters
- dd to delete the whole line
- a number and then 'dd' to delete a number of lines
- 'o' to start a new line directly below
- 'yy' to copy (or 'yank') the line
- a number and then 'yy' to copy a number of lines
- 'p' to paste the line/s you have just copied
- '%s/xxx/yyy/g' to find all instances of 'xxx' in the file and replace them with 'yyy'
- 'ZZ' to save and exit
- ':w' to save
- ':wq' to save and exit

These are the most useful commands, but the vi editor has a lot more functionality built-in. A 'cheat-sheet' with additional command options will be provided separately.

4 Shell scripting

A shell script is basically a text file with a list of unix commands that are executed sequentially. For example, you could use a shell script to automatically set up, run and restart molecular dynamics trajectories, or a series of quantum chemistry calculations at different levels of theory, or to map out a potential energy surface by systematically changing molecular coordinates (bond lengths, angles, dihedrals).

Exercise: You will be provided with a pdb file for the molecule glycylglycylglycine (GGG). Open this molecule using molecular visualisation software (e.g. Avogadro) and take a screen shot. Manually convert this into z-matrix format - specifying molecular structure by bond lengths, angles and dihedral angles (for rotations about bonds) using the OpenBabel program:

```
babel -ipdb GGG.pdb -ogzmat GGG.gzmat
```

Make two copies of this file:

```
cp GGG.gzmat GGG_cp1.gzmat
cp GGG.gzmat GGG_cp2.gzmat
```

Edit your new copied file manually to change one of the dihedral angles (d10) to a different value, then convert back to pdb format:

```
babel -igzmat GGG_cp1.gzmat -opdb GGG_cp1.pdb
```

Make the same change to your second copy of the file without requiring manually editing using:

```
sed -i 's/xxx/yyy/g' GGG_cp2.gzmat
```

where 'xxx' is what you want to find in the file (*e.g.* d10= 180.67) and 'yyy' is what you want to replace it with.

Convert GGG_cp2.gzmat back to pdb format using same procedure as for GGG_cp1 above. Check that GGG_cp1.pdb and GGG_cp2.pdb contain the same final geometries that are different to the original by opening these pdb files in your molecule builder/viewer program.

Create a shell script to do all these tasks sequentially by opening a new text file and entering the appropriate set of commands.

Now, it's easy to generate a series of pdb files with different d10 dihedral angles by repeating this process: copy-paste the original set of commands in the shell script, changing the file names and angles as you go.

Finally, make your newly created script executable (assuming you've called it auto_change_dihedrals.script) by typing the following command:

```
chmod +x auto_change_dihedrals.script
```

Execute your new script by typing:

```
./auto_change_dihedrals.script
```

Check to see that you have made your modified pdb files appropriately by opening them in your molecular visualisation software.

Extension: Think about how you could extend/modify this shell script to automatically produce input files for the quantum chemistry or molecular dynamics software you usually use. What changes, extensions and/or modifications are required? Hint: for quantum chemistry packages, converting back to pdb format is not required, but making additional changes to the gzmat file (converting to appropriate format for your software of choice, entering job control instructions e.g. level of theory, basis set) will be. For MD simulations, additional steps will be required after conversion back to pdb, generating force field parameter and molecular connectivity files, *etc.*. But all of these things can be automated using shell scripts.

5 Python scripting (extension)

Using shell scripts to automate your work is already pretty cool and potentially very time saving. But wouldn't it be even better if you could just tell the computer what dihedral you wanted to change, what increments you wanted, and let it do the rest for you? And to be able to change more than one dihedral at a time?

Python is a powerful and flexible scripting language that gives you the ability to do all of these things (and more!). In the interests of time, you will be provided with an example python script that automatically generates a shell script like the one you have just done manually but allowing you to change up to 3 dihedrals at a time. Have a look and see if you can follow what is happening. If you feel confident, have a go at writing your own python script using this as a template. You can also run this code.

Possible ways to extend or modify the script could be:

- Allow for any number, N, of dihedrals to be modified simultaneously. Hint: the code in the template has a number of nested loops in order to be able to handle up to three dihedrals at a time. This method is unfeasible in the case of N dihedrals. The function `all_combinations` may be useful to overcome this.
- Instead of just generating input files, try minimising the generated structure using a forcefield before saving the input file. The command `obminimize -ff UFF filename.extension` will be useful here. You will need to extract the minimised conformation from the output of the command and save it to a file. The function `extract_results` shows an example of how to extract results from a command. This command uses the Universal Force Field (UFF). Other forcefields available are: General Amber Force Field (GAFF) and MMFF94 force field (MMFF94).
- Perhaps you're not interested in performing QM calculations, and just want to plot the energy of a dihedral as it is rotated about (it should give a nice sinusoidal plot). Plotting example functions are provided. You will need to extract the energy of the conformation from the obminimize output. Hint: obminimize takes an option '-n' which gives an upper limit to the number of steps to take during minimisation. How about plotting a 2-dimensional version when two dihedrals are changed?
- The rotational energy profile of a dihedral should be sinusoidal. Why not calculate the sine (or cosine) function that matches it the best? How about fitting a Fourier series? What about fitting to more than one dihedral simultaneously?

Codeacademy has a really great beginners tutorial if you are interested in learning more python:

http://www.codeacademy.com/tracks/python

And I also recommend the text/reference book 'Practical Programming: An Introduction To Computer Science Using Python 3':

http://pragprog.com/book/gwpy2/practical-programming