UNIX for Chem 205

The vast majority of file moving, renaming and deleting can be done using your FTP client, but it will be helpful to have a basic grasp of UNIX commands.

General features of UNIX commands:

- A UNIX command consists of the command name, a series of modifiers and one or more arguments. Some commands don't take an argument and some commands don't need modifiers.
- Let's take the file removal utility (`rm`) as an example. This command requires an argument: the file to be removed. So the command `rm test.txt` will remove the file "text.txt" from your current directory. If we want to delete a folder and every file within the folder, we can use the modifier `-r`. The command `rm -r temp*` would remove every file or directory beginning with "temp" and delete every file *within* every directory beginning with "temp."
- Now let's consider `ls`, the command to list all of the items in your current directory. This command does not require an argument, but it will take modifiers. The modifier `-l` gives the long format for displaying files, including size and modification time. The modifier `-t` sorts the contents by time and `-r` reverses the sort order. Hence, `ls -l` would display all of the information for files in your current directory, sorted alphabetically, and `ls -lrt` would display all of the information for files in your current directory, reverse-sorted by modification time. This is equivalent to `ls -l -r -t`.

Directories:

- List contents of current directory: `ls`
- Change directory: `cd [directory name]`
    - `cd` with no argument returns you to your home directory
    - `cd ..` moves up one directory
- Create a directory: `mkdir [directory name]`
- Delete a directory and its contents: `rm -r [directory name]`

Files:

- Quickly scan through a text file: `more [file name]`
- Rename a file: `mv [file name] [new name]`
- Move a file: `mv [file path] [new path]`
- Copy a file: `cp [file name/path] [file name/path]`
- Edit a file: `vi [file name]`
- Delete a file: `rm [file name]`
    - * is the wildcard, so if you want to delete all files ending with ".chk" in a directory, you could type `rm *.chk`

Jobs:

- Submit a job: `bsub –q [queue] g09 [input file]`
    - Read about queues: `man queues`
- See what jobs you are currently running: `bjobs`
- Kill a job: `bkill [job ID]`

Get help:

- Read the manual entry for a command: `man [command name]`

Gaussian for Chem 205

The first step in creating a Gaussian input file is to draw your molecule in GaussView. Then save the file and open it in a text editor. You should see something like this:

```
%chk=ddf-ii-28-3.chk
%mem=50MW
%nprocshared=1
# opt freq rb3lyp/6-31g(d)

ddf-ii-28-3

0 1
 C
 C                  1             B1
 C                  2             B2    1             A1
 H                  1             B3    2             A2    3             D1
 H                  3             B4    2             A3    1             D2
 Cl                 1             B5    2             A4    3             D3
[and many more lines]
```

GaussView will often make strange guesses about what settings you want to use, so you should use the above settings as a template for your calculations.

Let's examine it line by line.

- The first line gives the name of the checkpoint file. The checkpoint file is a machine-readable record of the calculation. This will be important in restarting optimizations, performing new calculations on an optimized structure, or in performing transition state searches.
- The second line specifies how much memory should be allocated for your job.
- The third line tells Gaussian how many processors it has at its disposal. For the duration of Chem 205, we will be doing single-processor jobs only.
- The fourth line is the route section. It tells Gaussian what kind of calculation to perform. The "opt" keyword specifies a geometry optimization and "freq" requests a calculation of the vibrational frequencies of the molecule. After the job type is specified, we provide the level of theory: RB3LYP/6-31G(d). The "R" before B3LYP signifies that this is to be a "restricted" calculation of a closed shell configuration.
- After a blank line, we can give the job a name.
- After one more blank line, we get into the molecule specifications. "0 1" signifies an uncharged molecule in the singlet spin state.
- The remaining lines are the molecular coordinates in Z-matrix format. For more information on Z-matrices, read the Wikipedia entry for "Z-matrix (chemistry)".

Now suppose we want to find a transition state. A good approach is to draw something that you think is close to the transition state, fix a key bond length, angle or dihedral and optimize all other degrees of freedom. For example, if you are forming a bond, you could fix the forming bond to prevent your structure from collapsing to the starting materials or the product. The route section should look like this:

```
# opt=modredundant freq rb3lyp/6-31g(d)
```

where the "modredundant" option tells the optimization algorithm to constrain some degrees of freedom that you will specify. You list these at the very end of the input file. It should look like this:

```
[the rest of the input file]
   D72         -179.82853927
   D73          179.76221048
   D74           -1.61099651
[blank line]
7 11 F
[blank line]
[blank line]
```

Here we have specified a constraint: that the bond between atoms 7 and 11 should be frozen during the optimization. The blank lines at the end of the file are necessary for the file to be read properly. Once this job finishes, download the output file to your computer and open it with GaussView. Check for an imaginary vibrational frequency and see if it corresponds to the motion you expect.

If it does, save your modredundant-optimized structure as a new input file and change the header to look like this:

```
%chk=ddf-ii-28-3.chk
%mem=50MW
%nproc=1
# opt=(ts,readfc,noeigentest) freq rb3lyp/6-31g(d)

ddf-ii-28-3

[charge and multiplicity]
[z-matrix]
[blank line]
[blank line]
[blank line]
```

Make sure that the checkpoint file has the same name as the one you used in the constrained optimization and make sure you run this job from the same folder as you ran the first job from. We also have some new modifiers for the opt keyword. The "ts" modifier tells the optimization to look for a first order saddle point, "readfc" tells the optimization to get its second derivatives from the checkpoint file and "noeigentest" tells the optimization not to give up if it encounters a point with more than one imaginary frequency.

If your calculations are taking a prohibitively long time to run on one core, you can speed things up by running your job across multiple cores. A typical input header would look something like this:

```
%chk=ddf-ii-28-3.chk
%mem=200MW
%nprocshared=4
# opt freq rb3lyp/6-31g(d)
```

50MW/core is a pretty reasonable amount.

To submit this kind of job your `bsub` command should be something like this:

```
bsub –n 4 –q normal_serial –R span[ptile=4] g09 input.gjf
```

The –n modifier tells LSF how many cores to assign and the –R modifier tells LSF to make sure that all four cores are on the same node. This will work up to 8 cores, which is number of cores on the majority of nodes in Odyssey. If you want to run your job on cores across multiple nodes, you will need to use TCP/Linda.

If you want to learn about any of these commands, please refer to the manual pages for Gaussian 09 at http://www.gaussian.com/g_tech/g_ur/g09help.htm. This will be a valuable resource for you as your calculations grow more complex.