Codes to simplify the life of a structural/computational biologist

June 5, 2013

At some point in the life of a computer scientist, we must ask ourselves this question. Should we spend time writing codes to streamline a process that can be reused, or should we stick to copying and pasting code, making small changes everything time we want to work on something new. For a computer scientist, the answer is a resounding yes, but for researchers like ourselves, the effort it takes to write a comprehensive code that takes into account all options, variations of data, location of file etc. for a process we rarely use may not be worth it at all. This is especially true when there is always some deadline to be met. To that, I say, find a balance/compromise.

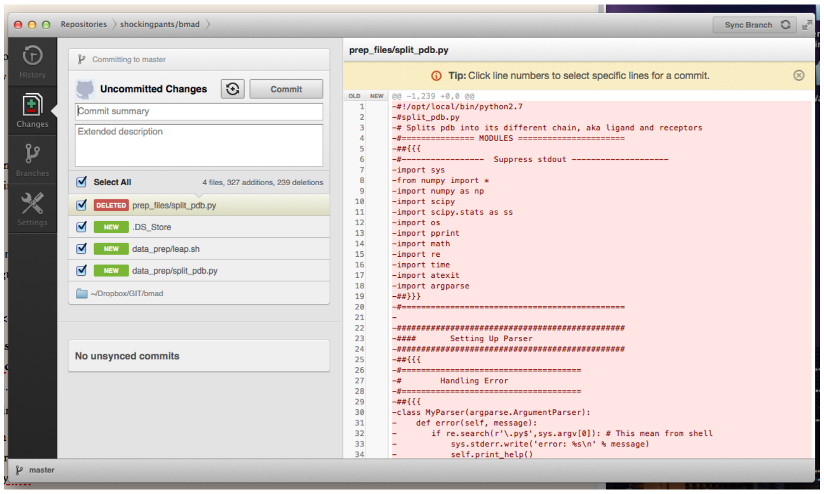
1 Using GITHUB

1.1 What is Git

- Git is a distributed version control and source code management system (from wiki)
- It transforms a working directory into a full fledged repository with complete history and full version tracking capabilities.

1.2 Installing git

1. Install git - http://git-scm.com/
2. Install git hub - https://github.com/
3. Follow the instructions. type **git** into the shell. See if it works.



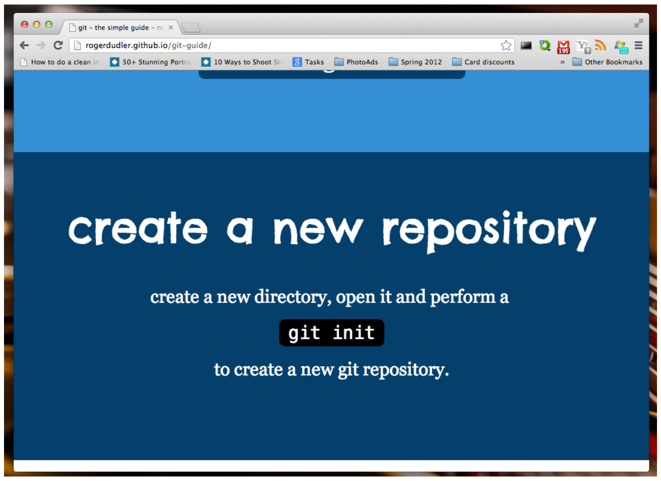
```
@@ -1,239 +0,0 @@
-#!/opt/local/bin/python2.7
-#split_pdb.py
-# Splits pdb into its different chain, aka ligand and receptors
-#============== MODULES =====================
-##{{{
-#-----------------  Suppress stdout -------------------
-import sys
-from numpy import *
-import numpy as np
-import scipy
-import scipy.stats as ss
-import os
-import pprint
-import math
-import re
-import time
-import atexit
-import argparse
-##}}}
-#===============================================
-
-###############################################
-####       Setting Up Parser
-###############################################
-##{{{
-#=====================================
-#        Handling Error
-#=====================================
-##{{{
-class MyParser(argparse.ArgumentParser):
-    def error(self, message):
-        if re.search(r'\.py$',sys.argv[0]): # This mean from shell
-            sys.stderr.write('error: %s\n' % message)
-            self.print_help()
```

Figure 1: git hub

4.

1.3 Learning and using git

1. http://git-scm.com/documentation
2. http://rogerdudler.github.io/git-guide/ (Fun and easy guide)



create a new repository

create a new directory, open it and perform a

```
git init
```

to create a new git repository.

Figure 2:

3.
4. Understanding 3 parts of git
   (a) Work Dir <-> Staging area <-> Git directory

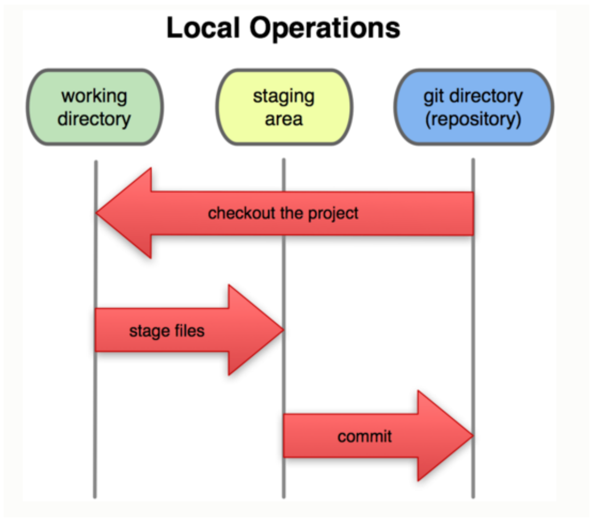


Figure 3: Working directory, staging area and git directory

5. **git init # use when in folder. This creates a repository**
6. **git clone https://github.com/shockingpants/bmad.git**
7. **git add [file]** # Adds file to stage. Files are tracked as well
   (a) Always add files to the staging area before commiting, unless you use commit -a (see below)
8. **git rm [file]** # Removes file from stage

9. **git status** # Checks which files are staged and which are not
10. **git commit** # Backup file # Everytime
    (a) **git commit -a** # Add previously tracked file automatically and commits them.
11. **git reset –hard HEAD^** # Returns to previous commit. HEAD^ points to previous commit. HEAD^^ points to previous previous commits
12. **git log** # Check history of commits
13. **git pull** # Pull file from repos again.

2 Preparing python

2.1 Why Python

- It is an interpreted language with a strong emphasis on code readability.
- It is **FREE**
- There are many free packages available for use.
- With C/C++ as its backend, you have the option to optimize the code for speed as well.
- You get to use structured progamming, which helps keep your codes organized.
- Googling for 'Python xxxxxx' is easier than 'R xxxx'

2.2 Version

- I stick with python2.7 since it is the version with the greatest third party software/module support. However, many people are migrating to python3.x so if you want to use the latest, you may wanna upgrade accordingly.
- The good thing about python is that it is an interpreted language that doesn't require compiling.

2.3 Packages

Just like R, you will need certain modules or packages for python to work. My recommended packages are

1. numpy - For array and data manipulation
2. scipy - For stats and other math packages
3. matplotlib - Toolbox for all kinds of 2D plotting and some 3D plotting. If you are are familiar with R, you are probably in good hands already. The syntax is very similar to MATLAB's handling of graphics.
4. biopython - For working with neucleotide, sequencing, alignment, fasta, pdb etc.
5. R-py2 - For integrating R into python. Not recommended unless you are trying to write everything in one language.

2.4 Compiling

Most software you come across will require some form of compiling. Fortran and C are the more popular codes around so it is good to have these compilers installed on your machine.

2.5 Permission

To run scripts from the terminal, you will need to change permission typically. To change permission, use **chmod 777 script.py.** For more information, please refer to http://en.wikipedia.org/wiki/Chmod.

2.6 Start up script

- To see hidden files from terminal, use **ls -a**
- Files with a period in front are usually hidden by default. E.g .bashrc
- To configure your python at start up, you will want to edit the following files: (type **env** to see if they have already been set)

  1. ~/.bashrc or whichever startup file for the shell/terminal you are using

     (a)
     ```
     #Example
     export PYTHONPATH=/usr/local/
     export PYTHONSTARTUP=~/.python27rc
     ```
     (b) PYTHONPATH contains the path from which python will look for modules, scripts, executables etc

     (c) PYTHONSTARTUP is the path to the start up script. Typically, you want to load commonly used packages during start up EG.

     (d)
     ```
     #!/usr/bin/env bash
     # ~/.python27rc
     # This is a startup script for python

     print 'GREETINGS, welcome to python.'
     import matplotlib.pyplot as plt
     import numpy as np

     # Plot Sthg cool everytime
     x=np.linspace(-2,2,1000)
     y1=np.sqrt(1-(abs(x)-1)**2)
     y2=-2*np.sqrt(1-(abs(x)/2)**0.5)
     plt.ion() #Allow interactive plot session
     plt.fill_between(x,y1,color='red')
     plt.fill_between(x,y2,color='red')
     plt.text(0, -0.4, time.asctime(), \
     fontsize=24, fontweight='bold', color='white', horizontalalignment='center')
     plt.show()
     time.sleep(1)
     plt.close('all')
     ```
     (e) Notice that the first line of the script contains #! (http://en.wikipedia.org/wiki/Shebang_(Unix)) followed immediately, without space, by a path pointing to python. This first line tells bash where to find the program to execute the rest of the script.

3 Data Preparation

3.1 General comments

3.1.1 Index

Indexing residues is a very tricky problem that everyone should be aware of especially when scripting. Imagine having to compare information between sequences from different species that do not align exactly due to insertion/deletion. Imagine having to compare peptides that are cap and not capped.

Sometimes, if there are multiple peptides in the system, the residue index will be repeated especially if the peptides both happen to be N-terminus chain. Imagine having to reference an index shared by the peptides!

In the pdb, you typically have **atom index** and **residue index**. These indexes, if they are pulled from rcsb, uniprot etc., typically reflect the AA actual position in the primary sequence of its parent proteins. However, when you extract a pdb from a trajectory using ptraj or cpptraj for example, the indexes get shifted and the first residue is now indexed as 1. To add another layer of complexity, most programming languages and software, such as vmd, ptraj etc., reference the first element in a data array as 0. And then there're caps..

My way of dealing with this is to keep a record of the original res index and shifting it so that the pdb index starts from zero. I would perform the shift after adding caps since I'll be working with capped proteins most of the time.

3.2 Split PDB

When preparing for MMPBSA, we typically need to separate a pdb file into separate receptor and ligand pdb files. split_pdb.py (First written by Yun, modified by Jon) will split the pdb file into separate files based on the chain letter.

```
#Usage
#From bash
split_pdb.py prot.pdb -o rec.pdb, lig.pdb #splits pdb into designated filenames.
#number of output file must match number of chain
split_pdb.py prot.pdb    #Automatically splits file with generic name
split_pdb.py prot.pdb -n #Check number of chains
split_pdb.py prot.pdb -h #help, usage details
```

Note: You will need to make sure that there is a different chain letter assigned to each peptide. See below

3.3 Assign chains

I don't have a script for this, but it can be done easily in pymol, albeit not automatically. I'll outline the steps as follow:

```
pymol prot.pdb #Loads protein
#Assuming you have loaded pymol successfully
set seq_view, on #Or you can click on 's' on the bottom right hand side
#Manually select residues you want to change the chain letter of.
#The selected residues will be highlighted
alter sele, chain='A'
#Select the second chain
alter sele, chain='B'
save prot_new.pdb, all
```

3.4 Assign resi

```
pymol prot.pdb #Loads protein
#Assuming you have loaded pymol successfully
set seq_view, on #Or you can click on 's' on the bottom right hand side
#Manually select residues you want to change the chain letter of.
#Alternatively, to shift all, use all
#The selected residues will be highlighted
alter sele, resi=str(int(resi)+5) # Here, we add an offset of 5
#Alters the resi of all the residues
alter al, resi=str(int(resi)+2)
```

```
save prot_new.pdb, all
```

3.5 Mutagenesis

This creates a mutant automatically using pymol as the backend.

```
usage: mut.sh -i 1ycr.pdb -o prot_mut.pdb -n 24,25 -t TYR
```

3.6 Leap.sh

This is used to generate the inpcrd and parmtop file for amber MMPBSA/GBSA or just trajectories in general from protein.pdb. Depending on the types of residue, you may need to generate the relevant parameter files using antechamber and edit the code accordingly.

```
leap.sh -i prot.pdb
```

4 Visualization

There are quite a few visualization tools out there. Since I personally use pymol and VMD, I'll give a short tutorial on the shortcuts I know. Hopefully, others can contribute and add on to what I know.

4.1 VMD

4.1.1 About

- VMD is a molecular visualization program that specializes in viewing
- VMD scripts are typically written in tcl.
  - TCL tutorial can be found here *http://www.tcl.tk/man/tcl/tutorial/tcltutorial.html*
  - VMD TCL commands can be found here *http://www.ks.uiuc.edu/Research/vmd/current/ug/node119.html*
  - Unfortunately, not all the gui features are incorporated into tcl so you will have to click through to access it

4.1.2 Installation

1. http://www.ks.uiuc.edu/Research/vmd/

4.1.3 Start Up Script

1. The Startup Script should be located here on the computer (Create it if it does not exist) **~/.vmdrc**
   (a) Please remember to add the period (**.**) in front to make it a hidden file and place the file in your home folder (**~**)
   (b) My file can be found on github. It is a copy of sukharevlab's with my personal add ons.
   (c) Enhanced file by sukharevlab.
      i. http://www.life.umd.edu/biology/sukharevlab/download/vmd_scripts/vmd.rc
      ii. http://www.life.umd.edu/biology/sukharevlab/download/vmd_scripts/vmd_enhanced_startup_file.htm
2. Things that are useful in a startup script will include repetitive comments such as loading trajectory, changing representation.
3. Explanation of start up script code

(a)
```
# turn on lights 0 and 1
light 0 on
light 1 on
light 2 off
light 3 off

# position the stage and axes
axes location off
stage location off

# turn on menus and position them based on pixel location
menu main on
menu graphics on
after idle {
        menu tkcon on
}
menu main move 5 196
menu graphics move 5 455
menu tkcon move 800 750
display projection orthographic

#Creates a new selection term called [side] This selects the side chain.
atomselect macro side {sidechain or name CA}

#================================
# Default representation
#================================
# This changes the default representation every a trajectory is loaded
mol default style NewCartoon
mol default color Name
mol default selection "all and not water and not hydrogen"
```

4.1.4 Mask and selection syntax

1. Useful Website

   (a) http://www.ks.uiuc.edu/Research/vmd/vmd-1.2/ug/vmdug_node137.html

```
name CA
resid 35
name CA and resname ALA
backbone
not protein
protein (backbone or name H)
name 'A 1'
name 'A *'
name "C.*"
mass < 5
numbonds = 2
abs(charge) > 1
x < 6 and x > 3
sqr(x-5)+sqr(y+4)+sqr(z) > sqr(5)
within 5 of name FE
protein within 5 of nucleic
same resname as (protein within 5 of nucleic)
protein sequence "C..C"
```

```
name eq $atomname
protein and @myselection
```

4.1.5 Commonly used codes

There are two main types of functions.
-The first is single letter commands that you press when the protein display window is active. (A)
-The second are functions that you use in the tcl/tk window. (B)

1. **g -** Side By Side Stereoscopic View (A)

   (a) To change this, open ~/.vmdrc and alter this part of the code

   (b)
   ```
   # Change from this
   user add key g {
           # Note that for SideBySide display, you will want to switch on the swap
           # command depending on whether you are more familiar with using the
           # cross or diverging eye technique.
           display stereo SideBySide
   }
   # to this (Change stereo to whichever is applicable to you)
   user add key g {
           display stereo ColumnInterleaved
   }
   ```

2. **loadparm -** loads parmtop from current path instead of having to click through the menu

   (a)
   ```
   proc loadparm { {parmtop prot_0wat.parmtop} } {
           ##{{{
           # Usage
           # loadparm prot_0wat.parmtop
           mol new $parmtop type "parm7"
           set name [file tail [pwd]]
           mol rename [molinfo top] $name
           ##}}}
   }
   ```

   (b) To get the "code" for the type of topology file, load it from the menu as usual and look for the code used in the command line window.

   (c) The parmtop will become the new TOP molecule

3. **loadtraj -** loads trajectory from current path

   (a)
   ```
   proc loadtraj { {traj prod_vac.nc} {skip 1} } {
           ##{{{
           # Usage loadtraj
           # Usage2 loadtraj prod_vac.nc 10
           # Skips determine the number of frames to skip

           mol addfile $traj waitfor "-1" step $skip
           ##}}}
   }
   ```

   (b) This loads trajectory into the top molecule

4. **s** - smooths all representation

(a)
```
proc s {{a 10}} {
        #==== Smooth ALL representations
        # Usage --> s 10     # This smooths over 10 frames
        # Get top molid
        set tp [molinfo top]
        # Get number of reps in top molecule
        set numrep [molinfo $tp get numreps]
        for {set i 0} {$i < $numrep} {incr i} {
                mol smoothrep $tp $i $a
                }
        puts "Smoothing␣[mol␣smoothrep␣$tp␣0]" }
```

5. **{.}** **{,}** **{/}** **-** Play trajectory forward, reverse, pause respectively.
6. **{+}** **{-}** **-** Speed up or Slow down the trajectory with smoothing applied. (To get plus sign, use shift =)
7. **lbl -** Labels target residue
   (a) Usage lbl index
      i. index starts from zero. Type resinfo to see which residue type corresponds to which index
      ii. The label shows resid, not the index, thus there is a discrepancy of 1.
   (b) To delete all labels, type **dl**
8. **hl** - Bolds bond depending on user selection

4.1.6 Preload trajectory

Here, I will describe briefly how you can customize your start up file and write a function to preload a system with the necessary representations. I'll use a combination of pseudo code and actual code to help you create your own.

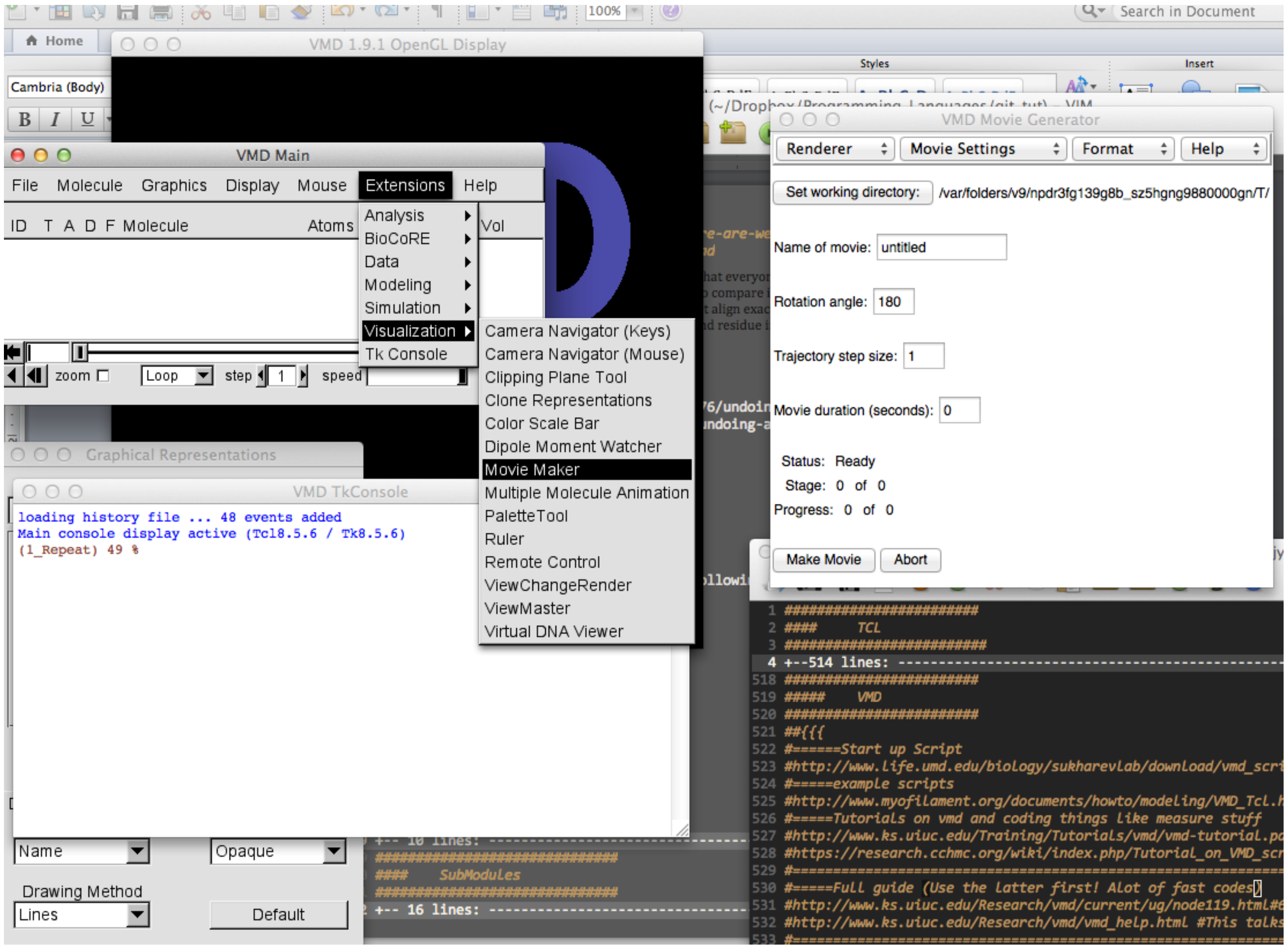
To automate the process, it is advisable that you keep all filenames for trajectory and topology the same so that you can set a default filename instead of referencing a new one everytime.

```
proc load_sample { {parmtop prot_0wat.parmtop} {traj prod_vac.nc} {skip n} } {
##{{{
#---------------------- Load files ------------------------------
# Checks for file existence.
if {[catch {loadparm $parmtop}]} {
        puts "$parmtop does not exist."
        }
if {[catch {loadtraj $traj $skip}]} {
        puts "$traj does not exist."
        }

#----------------------- Create Reps ---------------------------
#======== Delete current reps ===============
# This deletes all representations that were created when
d
#========== Get top molid ==============
# Get top molid. We will need this info to tell vmd
# which molecule to add the representations into.
# Here, $tp is the molid
set tp [molinfo top]
#============= Add representation ===============
####################
###### REP 0 #######
####################
mol addrep $tp
# Syntax ==> mol [command] [repid] [molid] arguments
mol modselect 0 $tp "all and no water"
mol mocolor 0 $tp Name
mol modstyle 0 $tp NewCartoon
####################
###### REP 1 #######
####################
mol addrep $tp
# Syntax ==> mol [command] [repid] [molid] arguments
mol modselect 1 $tp "water"
mol mocolor 1 $tp Name
mol modstyle 1 $tp NewCartoon

#--------------- Smooth trajectory ---------------------------
s
##}}}
}
```

4.1.7 Aligning proteins

4.1.8 Making Movies

1. Extension >> Visualization >> Movie Maker
2. Apply the following settings.
   (a) For smooth trajectory, make sure you apply smoothing before extracting the snapshot. If you are picking a snapshot every 10 frames, you will want smoothing to be at least 15 frames
   (b) Enlarge the window as wide as possible to get the largest resolution video you can manage.
   (c) Be sure to update the directory to a place you can locate easily.
3. For higher def videos, be sure to generate individual snapshots
   (a) Combine the individual snap shots using Time Lapse Assembler (Mac)
      i. http://mac.brothersoft.com/time-lapse-assembler-download.html
   (b) This generates a .mov file



```
loading history file ... 48 events added
Main console display active (Tcl8.5.6 / Tk8.5.6)
(1_Repeat) 49 %
```

/var/folders/v9/npdr3fg139g8b_sz5hgng9880000gn/T/

Name of movie: untitled

Rotation angle: 180

Trajectory step size: 1

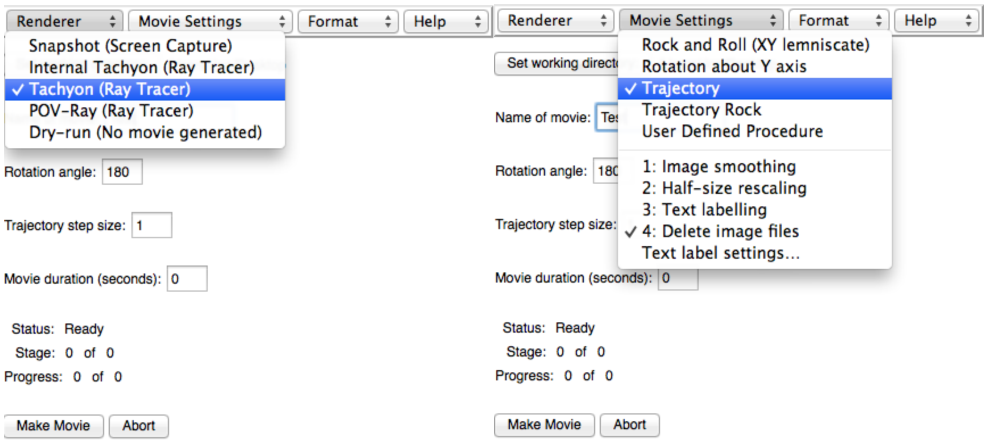
Movie duration (seconds): 0

Status: Ready
Stage: 0 of 0
Progress: 0 of 0

```
+-- 10 lines: ----------------------------
##############################
####     SubModules
##############################
+-- 16 lines: ----------------------------
```

```
########################
####      TCL
########################
+--514 lines: ---------------------------------------------------
########################
#####     VMD
########################
##{{{
#=====Start up Script
#http://www.life.umd.edu/biology/sukharevlab/download/vmd_scr
#=====example scripts
#http://www.myofilament.org/documents/howto/modeling/VMD_Tcl.h
#=====Tutorials on vmd and coding things like measure stuff
#http://www.ks.uiuc.edu/Training/Tutorials/vmd/vmd-tutorial.p
#https://research.cchmc.org/wiki/index.php/Tutorial_on_VMD_scr
#=========================================================
#=====Full guide (Use the latter first! Alot of fast codes
#http://www.ks.uiuc.edu/Research/vmd/current/ug/node119.html#
#http://www.ks.uiuc.edu/Research/vmd/vmd_help.html #This talks
```

Figure 4:



Name of movie: Tes

Rotation angle: 180

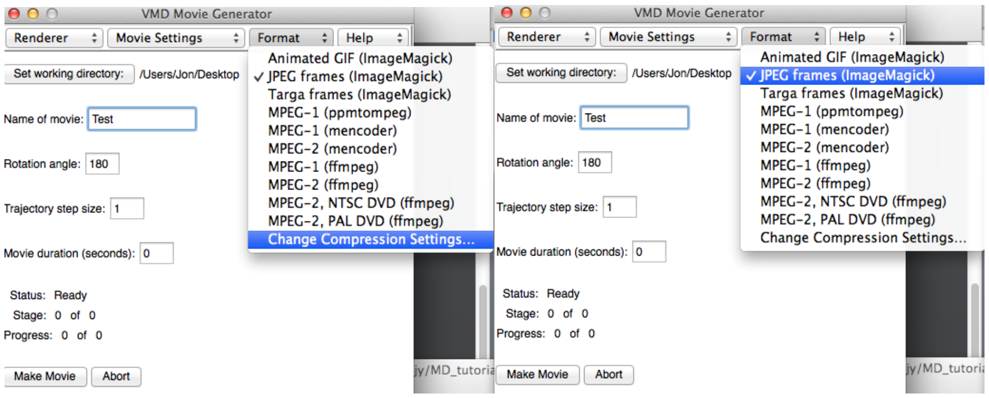
Trajectory step size: 1

Movie duration (seconds): 0

Status: Ready
Stage: 0 of 0
Progress: 0 of 0

Figure 5:

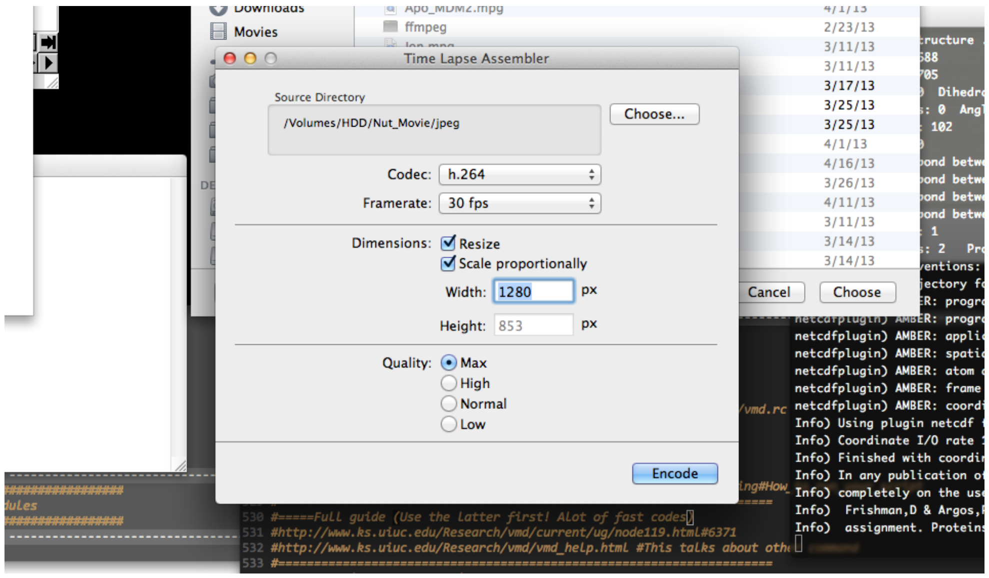
4.



Set working directory: /Users/Jon/Desktop
Name of movie: Test
Rotation angle: 180
Trajectory step size: 1
Movie duration (seconds): 0
Status: Ready
Stage: 0 of 0
Progress: 0 of 0

Figure 6:

6.



Source Directory
/Volumes/HDD/Nut_Movie/jpeg
Codec: h.264
Framerate: 30 fps
Dimensions: [✓ Resize]
[✓ Scale proportionally]
Width: 1280 px
Height: 853 px
Quality: [● Max] High Normal Low

```
#=====Full guide (Use the latter first! Alot of fast codes
#http://www.ks.uiuc.edu/Research/vmd/current/ug/node119.html#6371
#http://www.ks.uiuc.edu/Research/vmd/vmd_help.html #This talks about othe
#==================================================================
```

Figure 7:

7.

4.2 Pymol

4.2.1 Start Up Script

1. It can be found in **~/.pymolrc.**

```
# ~/.pymolrc
# Sets sequence to on always
set seq_view, on
```

4.2.2 Mask Syntax and Selection

1. Useful Sites

   (a) http://www.pymolwiki.org/index.php/Property_Selectors

   (b) http://www.pymolwiki.org/index.php/Selection_Macros

2. The more commonly used ones include

    (a) **name** - Eg. ca cb cg
    (b) **resn -** Eg. asp glu
    (c) **resi -** residue index
    (d) **chain**
    (e) **ss -** secondary structure Eg. h+s+l+ " "
    (f) **b -** B-factor

3. Examples

    (a) Usage: **select [name of selection output], [selection criteria]**
    (b) select backbone, name ca+c+n
    (c) select closest, lig within 4 of prot
    (d) select test, prot and chain b and resi 1-5

4.2.3 Aligning figure

You may either align two proteins straight away, or align selections that you have created

```
#Syntax of aligning A to B
align [A], [B]
#Align proteins
align prot1, prot2
#Align back bone of proteins
align prot1 and name ca+c+n+o, prot2 and name ca+c+n+o
```

4.2.4 Generating pdb with new beta factor

1. http://www.pymolwiki.org/index.php/Advanced_Coloring
2. Color b factor
3. **spectrum b, palette= ______, minimum=__________, maximum=____________**

4.2.5 Generating high quality figure

1. http://ihome.cuhk.edu.hk/~b102142/pymol/pymol_tutorial.html

5 Data Extraction

6 Data Analysis

6.1 Storing residues of interest

Sometimes, in the process of analyzing the data, you come across residues that happen to pique your interest. It may be closest residues, hydrophobic residues, residues exhibiting concerted motion etc. Here, I propose a way to save these selections so that you can pull them out when you need it with customizable seleciton syntax.

```
def dec(fn,offset=0,sep=', '):
        ##{{{
        """     Stores and retrive data of interest
        index is stored starting from 0.
        0 usually points to the cap ACE.        """
        a=fn()
        assert type(a)==list
        def new_fn(offset=0,sep=' ,'):
                dat=sep.join(map(lambda x: str(x+offset),a))
                return dat
        return new_fn
        ##}}}

@dec
def sec_site():
##{{{
"""     Within 4A       """
dat=[1,2,4,25,26,76,77,79,80,83,85]
return dat
##}}}

###############################
# Usage:
>>>sec_site()
'1 ,2 ,4 ,25 ,26 ,76 ,77 ,79 ,80 ,83 ,85'
>>>sec_site(sep='+')
'1+2+4+25+26+76+77+79+80+83+85'
>>>sec_site(sep=' and ',offset=10)
'11 and 12 and 14 and 35 and 36 and 86 and 87 and 89 and 90 and 93 and 95'
###############################
```