Pymol 学习笔记

转自：http://bbs.chinanmr.cn

整理：ChinaNMR.cn

Pymol 学习笔记（一）：简介&安装

Pymol 是一个开放源码，由使用者赞助的分子三维结构显示软件，由 Warren Lyford DeLano 编写，并且由 DeLano Scientific LLC 负责商业发行。

Pymol 被用来创作高品质的分子（特别是生物大分子如蛋白质）三维结构。据软件作者宣称，在所有正式发表的科学论文中的蛋白质结构图像中，有四分之一是使用 Pymol 来制作的。

Pymol 名字的来源：“Py”表示该软件基于 python 这个计算机语言，“Mol”则是英文分子（molucule）的缩写，表示该软件用来显示分子结构。

由于实验需要，本人正在学习该软件，在这里把学习过程记录下来，希望对有需要的朋友有所帮助。今天先来说说安装吧。

自 2006 年 8 月 1 日起，DeLano Scientific 对事先编译好的 PyMOL 执行程序（包括 beta 版）采取限定下载的措施。目前，只有付费用户可以取得。不过源代码目前还是可以免费下载，供使用者编译。如果你和我一样，不想为此花钱的话：

1. 如果你是 Windows 用户，首先下载 Pymol 的源代码。

   然后安装 CygWin，并且确保正确安装以下模块：

   - C++ (gcc or g++ package name)
   - Python
   - OpenGL
   - PNG

   然后在源代码目录里面依次运行：

2. 如果你是 Linux 用户，首先确保以下东东已安装：

   - Python
   - Pmw
   - OpenGL driver（我用的是 NVdia）
   - libpng
   - Subversion client（下载源代码需要）

   然后下载 Pymol 的源代码

```
$ mkdir pymol-src
$ svn co https://pymol.svn.sourceforge.net/svnroot/pymol/trunk/pymol pymol-src
```

然后进入源代码目录

```
# cd pymol-src
```

开始依次编译

```
# python setup.py install
# python setup2.py install
```

拷贝执行脚本到某个$PATH，安装就搞定了

```
# cp ./pymol /usr/bin
```

如果运行时得到错误信息"ImportError: No module named Pmw"，那么你应该运行

```
# python setup2.py install pmw
```

如果你在使用 Gentoo，请确保编译 python 时添加了 tcl/tk 支持，否则运行是会提示错误 "ImportError: No module named _tkinter"

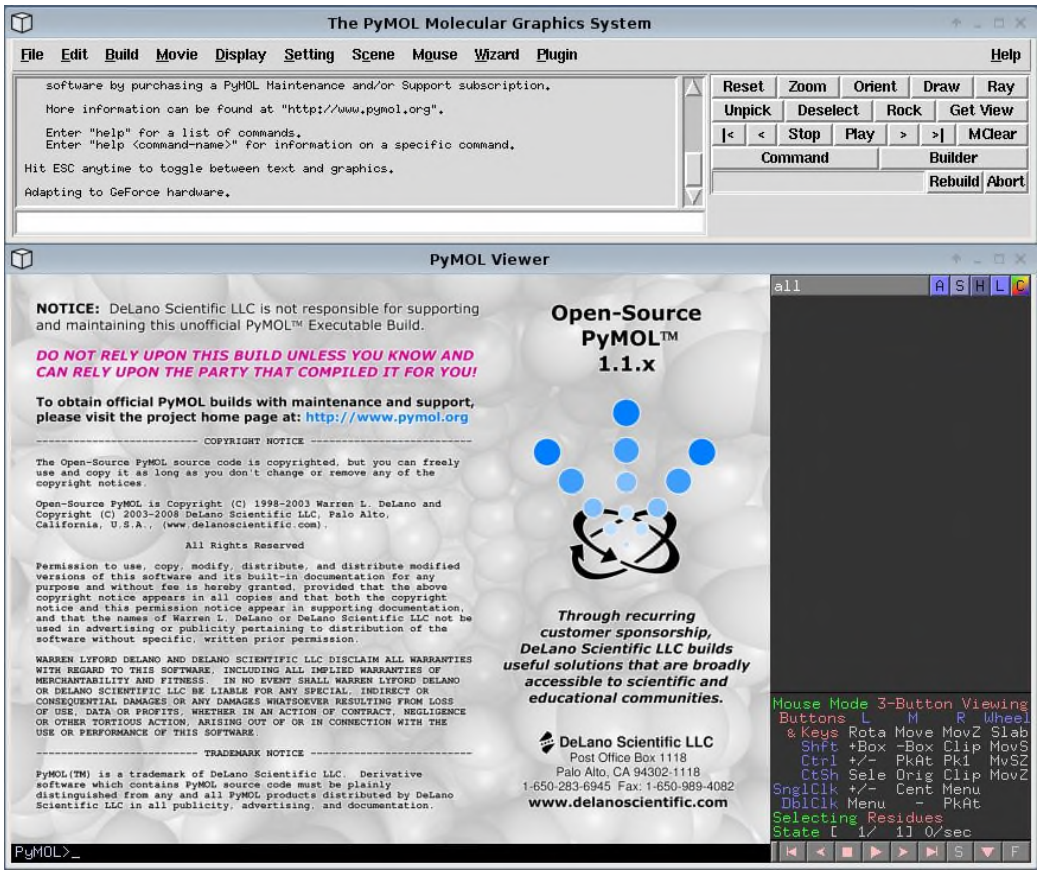
```
# USE="tcl tk" emerge python
```

好了，下面我们就可以进入 Pymol 的世界了。

Pymol 学习笔记（二）：基本的鼠标操作

这里主要介绍一下 Pymol 的基本操作，包括窗口菜单、加载文件、图像的基本鼠标操作等等。

当你打开 Pymol 后，你将会看到如下图所示的界面：



该界面分为 2 窗口，上面的外部 GUI 窗口（External GUI）和下面的 Viewer Window。Viewer Window 又分为左右两块，左边用来显示结构图像的（Viewer），右边则是一个内部 GUI 窗口（Internal GUI)。Viewer 自身包含一个命令行（如图中左下方的 PyMOL>提示符），可以用来输入 Pymol 命令；在 Inernal GUI 中则可以选定一些特定的对象并完成一些操作。External GUI 则包含一个标准菜单、一个输出区、一个命令行输入区以及右边的一些常用命令按钮。请注意，标准的“复制、剪切和粘贴”操作只能在 External GUI 中完成，并且必须使用“Ctrl+C、Ctrl+X 以及 Ctrl+V”来完成，这也是这个所谓的外部 GUI 的最重要的优点。

加载文件，有二种方法：

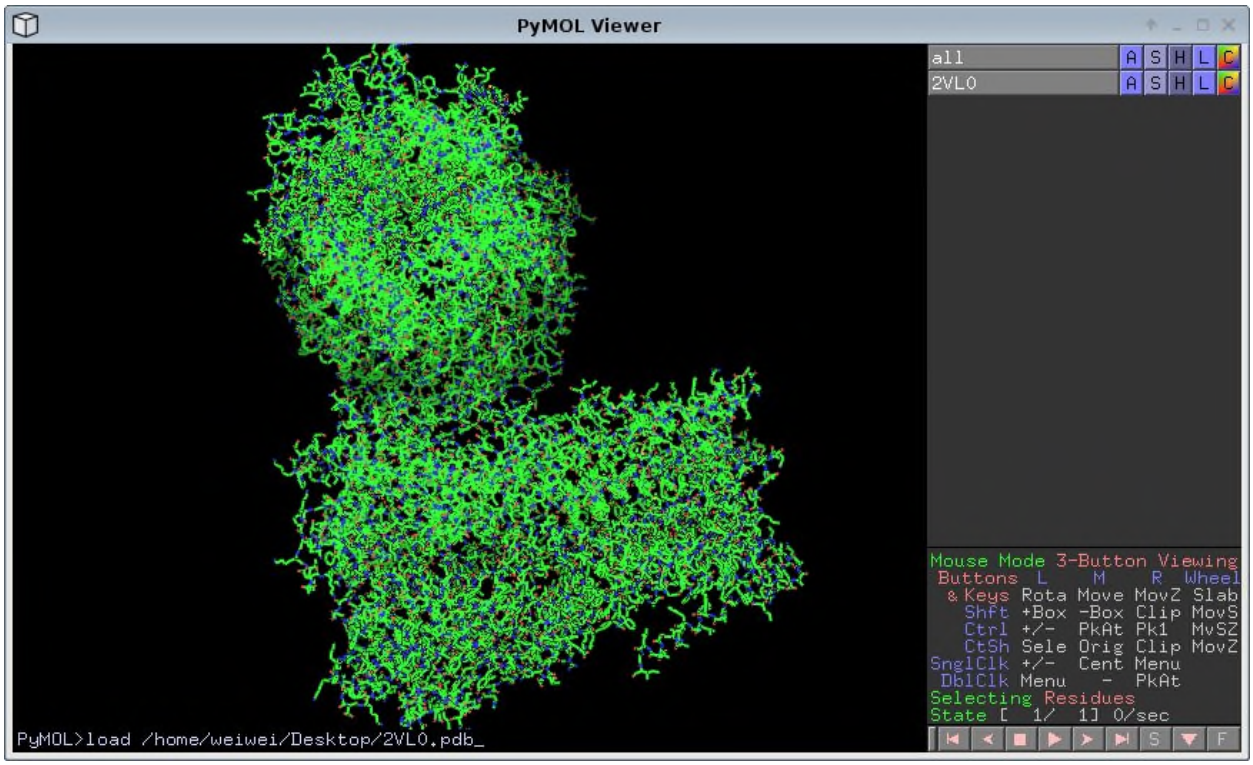
1. 在 External GUI 中选择 File － Open
2. 使用命令行：

```
load <filename>
```

例如我们现在从 www.pdb.org 上下载了一个离子通道蛋白的 pdb 文件（PENTAMERIC LIGAND GATED ION CHANNEL FROM ERWINIA CHRYSANTHEMI），名字为 2vl0.pdb，然后用 pymol 打开它：

```
load 2vl0
```

该蛋白质的结构就被显示出来啦，如下图：



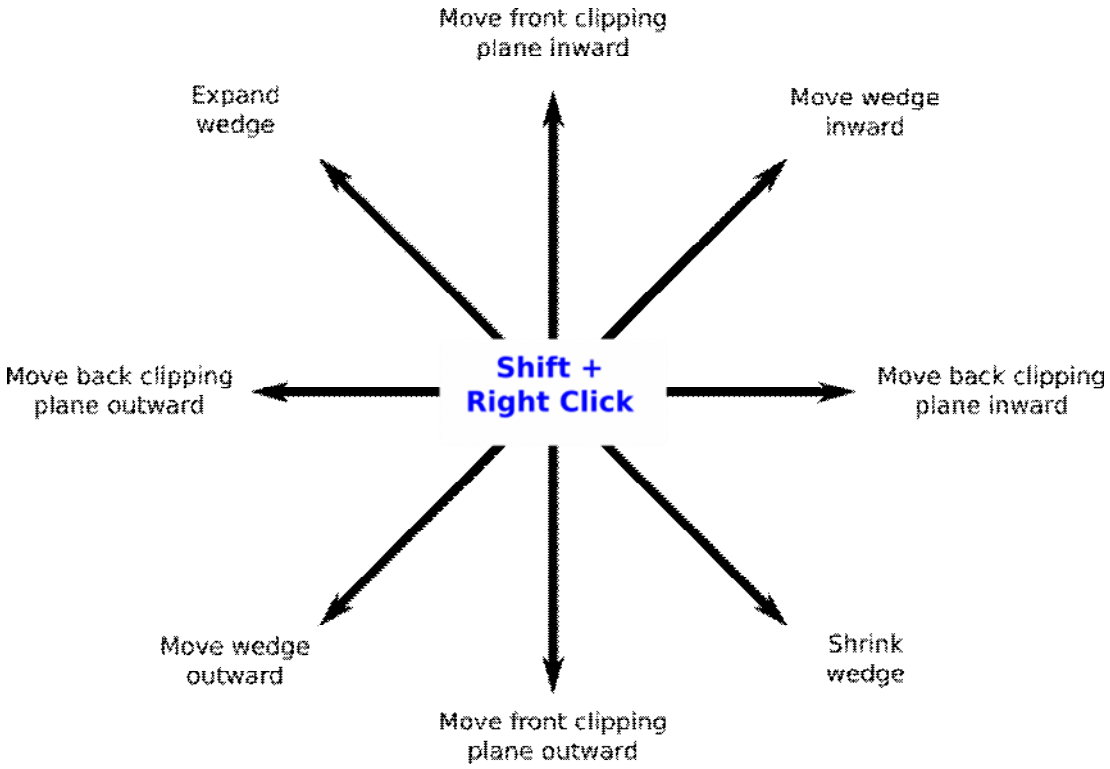
基本的图像操作:

是不是觉得上面的那个三维结构图看起来乱七八糟的阿，那是因为蛋白质分子都是由成千上万个原子组成的，而 Pymol 打开 pdb 文件时是默认把所有的原子都显示在那个小小的 Viewer 窗口里面的，当然看起来就很乱了。这时候就需要我们对这个图像进行一些操作，来得到漂亮的清晰的蛋白质三维结构图。

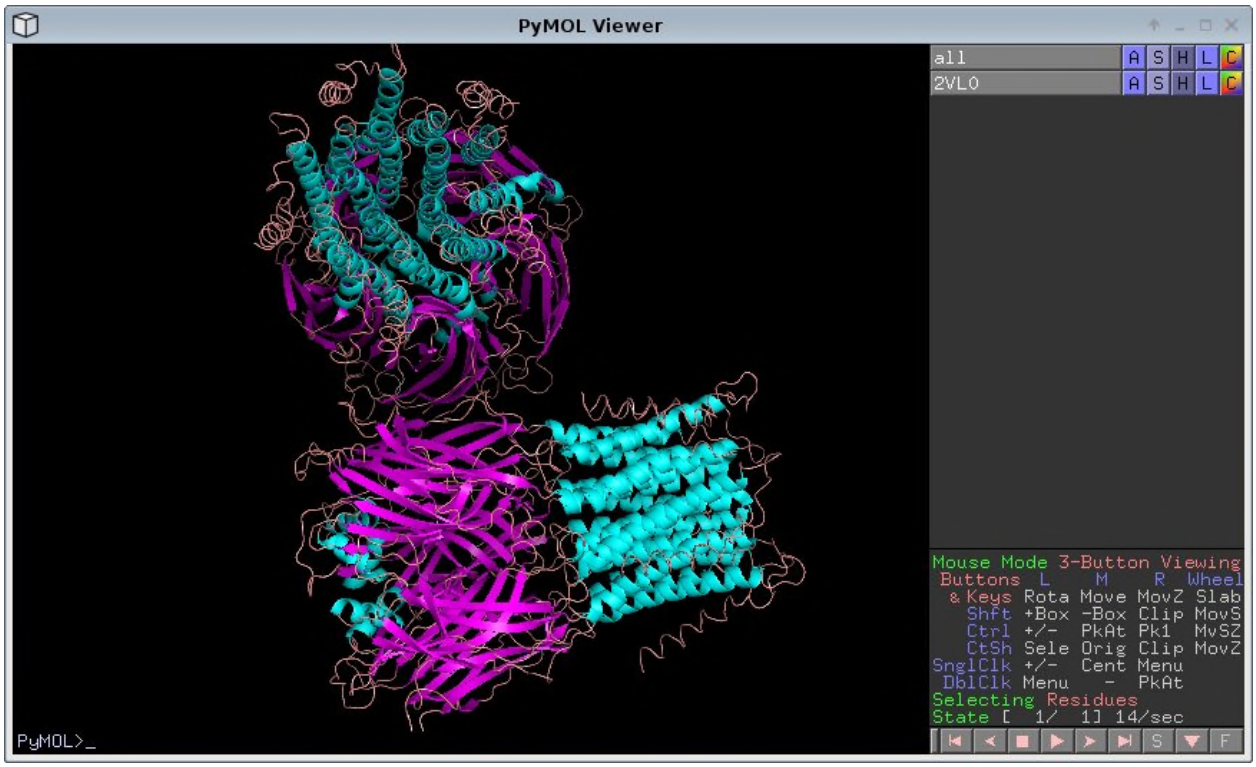
先说说鼠标吧。

- 任意旋转图像： 对准图像的任意处点住鼠标左键然后移动鼠标。
- 放大/缩小图像： 对准图像的任意处点住鼠标右键然后移动鼠标：向上是缩小，向下则是放大。
- 移动图像： 对准图像的任意处点住鼠标中键或者滚轮，然后移动鼠标。
- 设定图像旋转中心： Ctrl＋Shift＋鼠标中键或滚轮。
- 移动剪切平面： Shift＋鼠标右键。鼠标上下移动：调整前剪切平面（离你近的）；鼠标左右移动：调整后剪切平面（离你远的）。

最后一项“移动剪切平面”有点不容易理解，需要多试几次。配合下面的示意图你会发现 Pymol 的这项设定其实很方便。



今天没时间了，明天还要出远门，就学到这里吧，用下面这个图作为结束，其实就是用 cartoon 的形式显示了上面的那个蛋白质，不过还比较难看。。。



Pymol 学习笔记（三）：基础 Pymol 命令

这里主要介绍一下 Pymol 的一些基本命令操作。就像 Linux 一样，要想更好的操作 Pymol，掌握一些常用的命令是必不可少的。 Pymol 是区分大小写的，不过目前为止 Pymol 还是只用小写，所以记住，所有的命令都是使用小写字母的。

当你开始用 Pymol 来完成一个项目时，你也许想会让 Pymol 自动保存你所有输入过的命令，以方便日后你再次读取并修改。这个可以通过创建一个 log 文件来达到，该文件的后缀名应为.pml，记住，Pymol 像 Linux 一样支持 Tab 键命令补全：

```
Pymol> log_open log-file-name.pml
```

如果你想终止记录，只需要键入：

```
Pymol> log_close
```

好了，现在载入 pdb 文件（继续前用的 pdb 文件）：

```
Pymol> load 2vlo.pdb
```

现在 Pymol 就创建了一个叫 2vlo 的对象，你可以在内部 GUI 窗口里面看见这个项目的名字。但是你也可以自己定义该项目的名字（如 test）：

```
Pymol> load 2vlo.pdb, test
```

下面说说如何来操作你新建的对象。

首先：

```
Pymol> show representation
Pymol> hide representation
```

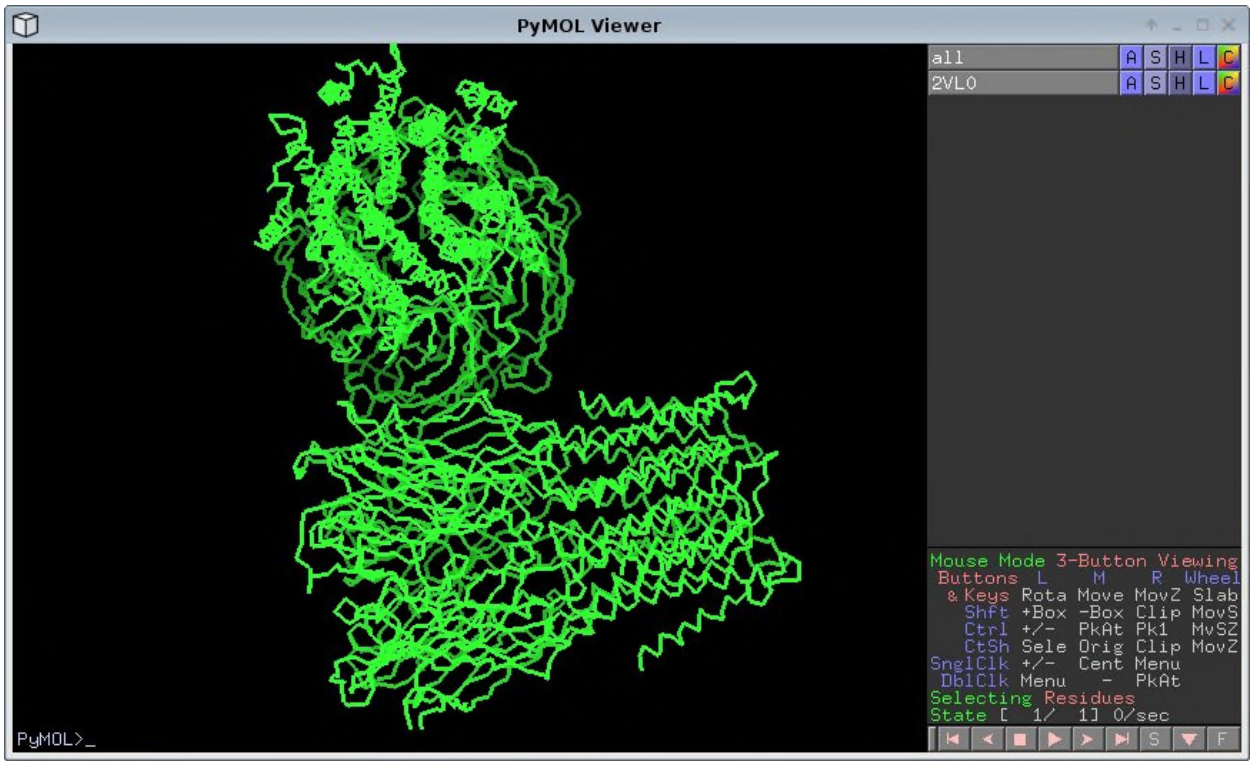
其中 *representation* 可以为：cartoon, ribbon, dots, spheres, surface 和 mesh。

使用这 2 个命令可以让 Pymol 以不同的方式显示蛋白质结构。

例如当我们键入：

```
Pymol> hide lines
Pymol> show ribbon
```

我们将得到如下结果：



也许你已经注意到结构中有 2 个一模一样的蛋白质分子，只是方向不同而已，那么如何让 Pymol 只显示当中的一个分子呢？首先输入如下命令：

```
Pymol> label all, chains
```

这个命令的作用是让 Pymol 给蛋白质结构中的“链”编号，你会发现，第一个分子由“链”A－E 组成，第二个则由 F－J 组成。

好了，如果我们想把一个蛋白质分子去掉，那么只要把“链”A－E 或者 F－J 去掉即可：

```
Pymol> hide ribbon, chain f+g+h+i+j
```

上面的东东还可以这样完成：

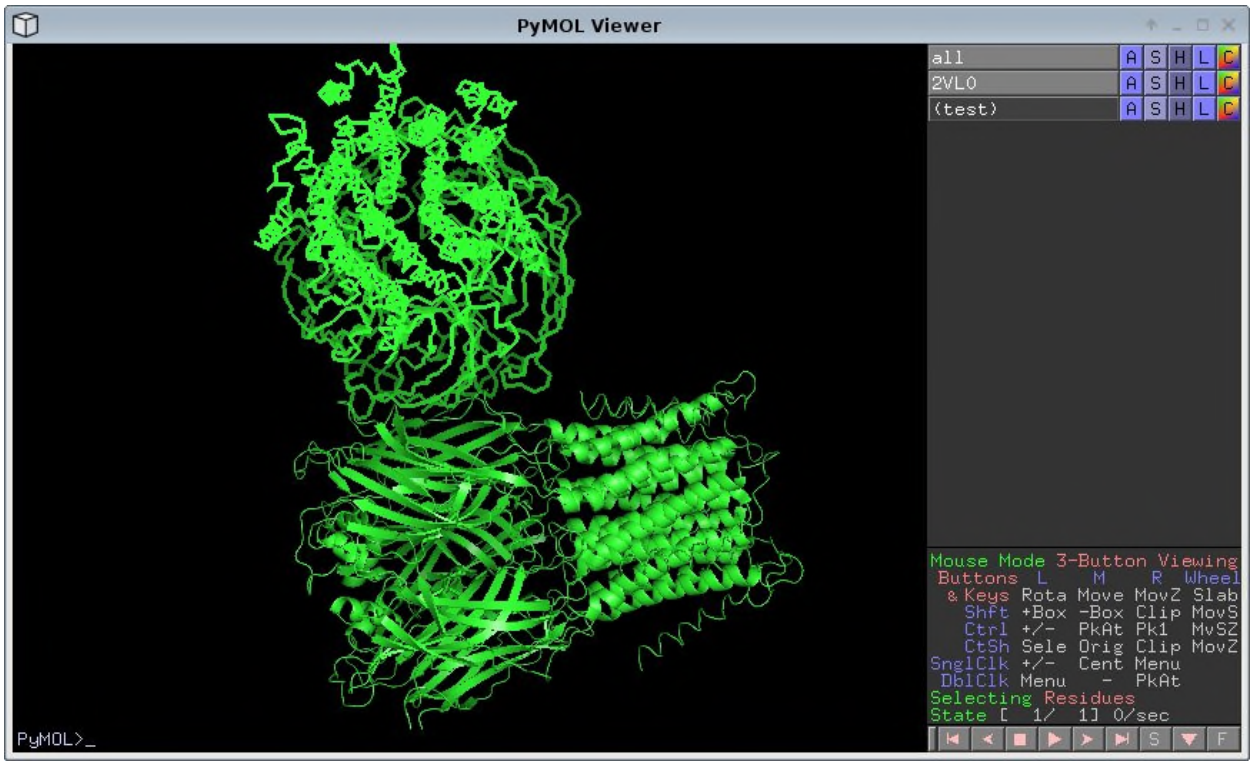
```
Pymol> select test, chain f+g+h+i+j
Pymol> hide ribbon, test
```

上面的第一句命令的作用是选择“链”F－J，并命名为 test，然后在第二句命令中隐藏它。这样做的好处是，一旦你选择并命名了某个目标，你可以在后面随时对它进行各种操作。并且你在右边的控制面板里面也可以看到你选定的目标，并可以对其进行操作。

比如你可以：

```
Pymol> hide everything, test
Pymol> show cartoon, test
```

这样你会得到：



说到这里就提到了 Pymol 的一个比较重要的东东，就是选择并命名目标，它的基本语法就是：

```
Pymol> select selection-name, selection-expression
```

其中名字可以由字母[A/a－Z/z]，数字[0－9]已经下划线[_]组成，但是要避免使用：

```
! @ # $ % ^ & * ( ) ' " [ ] { } \ | ~ ` < > ? /
```

如果你要删除你选定的目标或者整个对象，你可以：

```
Pymol> delete selection-name

Pymol> delete object-name
```

下面讲讲如何给对象以及目标改变颜色。预定义的颜色名字可以在外部 GUI 窗口的 Settings － Colors 中找到：

```
Pymol> color color-name

Pymol> color color-name, selection-expression
```

比如我们可以：

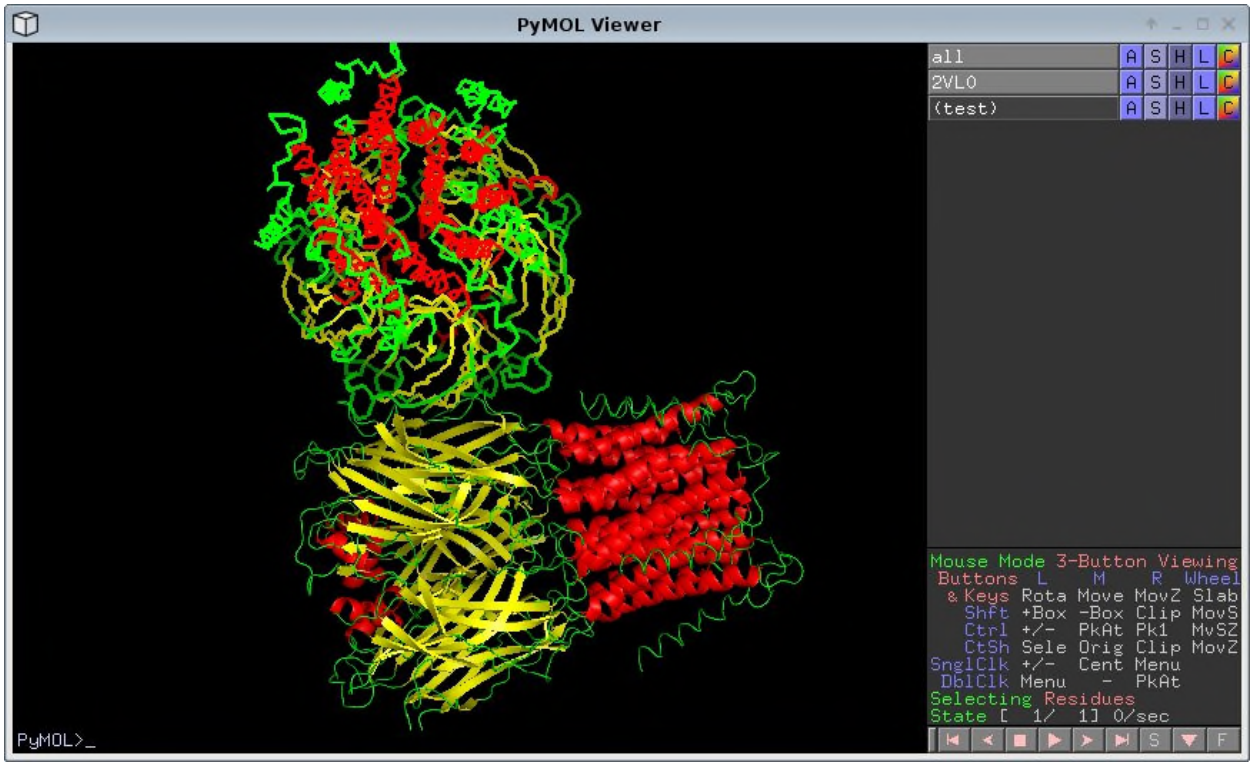
```
Pymol> color red, ss h

Pymol> color yellow, ss s

Pymol> color green, ss l+""
```

其中“ss”代表 secondary structure，“h”代表 Helix，“s”代表 Beta sheet，l+""代表 Loop 和所以其他结构。

这 3 句的作用分别是把所有的 Helix 变成红色；把所有的 Beta sheet 变成黄色；把所有的 Loop 以及其他部分变成绿色，于是我们得到：



Pymol 可以同时打开多个 pdb 文件：

```
Pymol> load object-name-1.pdb
Pymol> load object-name-2.pdb
```

如果你想暂时关闭/打开某个对象，可以这样：

```
Pymol> disable object-name-1
Pymol> enable object-name-1
```

你也可以用 disable 命令去除最后一个选择的目标上出现的粉红色的小点，但是该命令并不会使你选定的目标不可见。

```
Pymol> disable selection-name
```

使用鼠标通常是改变图像视角的最方便的办法，不过命令如 zoom，orient 等等有时候使用起来也是很有用的，它们提供了另一种改变图像视角的办法。

放大选定目标：

```
Pymol> zoom selection-name
```

定向选定目标，可以使选定目标最大的尺寸水平显示，次大的尺寸竖直显示：：

```
Pymol> orient selection-name
```

你也可以用 view 命令保存你目前的视角，注意，该命令只保存视角，并不保存你的对象显示方式：

```
Pymol> view key, action
```

其中“key”是你随便给当前视角定的名字，“action”可以为：store 或者 recall。如果不加任何“action”，则默认为 recall：

```
Pymol> view v1, store
Pymol> view v1, recall
Pymol> view v1
```

说了这么多，最后说说如何保存文件吧。Pymol 有 3 个层面的保存方式，下面来分别说说。

1. 使用 log_open 命令把你所有使用过的命令记录为一个文本文档：

   ```
   Pymol> log_open script-file-name
   ```

   这样以后当你再次调用该文档时，Pymol 将执行上面的所有命令：

   ```
   Pymol> @script-file-name
   ```

   不过注意，如果你想记录当前视角，则必须使用 get_view 命令。

   你可以选择外部 GUI 窗口中的 File － Append/Resume/Close Log 来分别暂停记录/恢复记录/停止记录该文档。

   你可以随时编辑该文档。

   在 linux 下，该文档的默认保存目录为当前用户的 home 目录。

2. 如果你想下次打开 Pymol 时直接回到当前所在的状态，那么你可以选择外部 GUI 窗口里面的 File － Save Session，创建一个会话文件（.pse）。

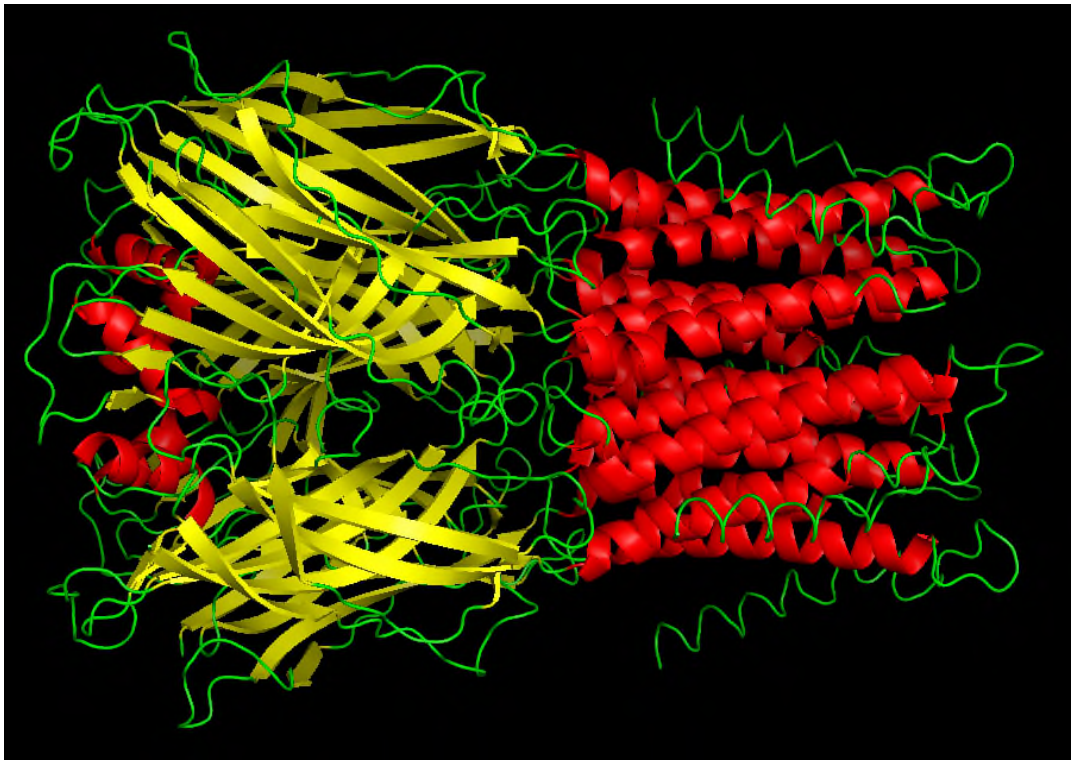
   该会话文件和上面提到的文档文件的区别在于，首先文档文件可以编辑，但会话文件不可以；记录文档文件前必须先运行 log_open 命令，而会话文件可以随时创建；最后文档文件以文档形式运行（@），而打开会话文件则必须选择外部 GUI 窗口中的 File － Open。

   什么时候需要创建会话文件呢？比如当你在某时有多个选择时，你可以保存当前状态，然后一一尝试这些选择，不满意时只需要重新打开该会话文件即可。也就是说创建会话文件起到了“undo”的作用，这正是 Pymol 所缺少的。希望开发者能赶快加入该功能，那么这个会话文件好像就没什么大用了，呵呵。

3. 如果你觉得当前显示窗口里面显示的结构图像已经满足你的要求了，你可以把它保存为图片。在这之前你可以使用 ray 命令来优化你的图像，它可以使你的图像具有三维的反射及阴影特效：

```
Pymol> ray
Pymol> pngyour_path/image_name
```

最后就用该命令导出的图片结束这次笔记吧。



Pymol 学习笔记（四）：Pymol 命令的语法与目标选择的表达

上次介绍一些 Pymol 的基本命令。现在来具体说说 Pymol 命令的语法，还有在选择操作目标应该如果表达。个人觉得这部分内容对学习 Pymol 来说是至关重要的。

从上次讲的一些例子中不难看出，Pymol 的命令都是由关键词（keyword）加上一些变量（argument）组成，格式如下：

```
Pymol> keyword argument
```

其中关键词（keyword）当然是必须的，而变量则不是必须的，比如退出命令 quit 就不需要附加变量：

```
Pymol> quit
```

当然更多的命令通常是需要加变量的，比如放大命令 zoom，但是你会发现即使你不加任何变量该命令也可以被执行，这是因为 Pymol 的许多命令有一个默认变量，下面两个命令的作用是一样的，其中的目标选择 all 就是 zoom 的默认变量：

```
Pymol> zoom
Pymol> zoom all
```

还有些命令可以带多个参数，比如加色命令 color，它的用法如下：

```
Pymol> color color-name
Pymol> color color-name, selection-expression
```

第一个 color 虽然只带一个变量"color-name"，但其实它包含了第二个默认变量 all，所以它的作用是把整个结构变成"color-name"的颜色。

第二个 color 带两个变量，和第一个的区别就是把默认的目标选择变量 all 变成了"selection-expression"，也就是说只有被这个变量选中的部分才会被变成"color-name"定义的颜色。要注意的是，如果一个命令带多个变量，则这些变量之间必须用逗号","隔开。

通过这个例子，大家可以发现，有些变量本身是很简单的，比如"color-name"，就是一个颜色名字而已，没什么复杂的。另一些则不一样，比如"selection-expression"，它可以很简单，也可以非常的复杂。这个东东，我称之为选择表达，对 Pymol 命令的使用非常重要，所以下面要详细的讲一下。

选择表达（selection-expression）表示的实际就是一些被选中的部分，它们可以是一些个原子，一些个 Helix，一些个 Beta sheet，或者它们的混合物。你可以给你的选择表达起个名字，以便可以多次使用它们。名字可以由大小写字母，数字以及下划线[_]组成，但是因避免使用下列符号：

```
! @ # $ % ^ & * ( ) ' " [ ] { } \ | ~ ` < > ? /
```

选择表达由所谓的"selector"加上"identifier"组成，其中"selector"定义了某类属性，而"identifier"则在该类属性下需要被选择的部分。如下例：

```
Pymol> select test, name c+o+n+ca
```

其中"name"就是一个 selector，它表示在 pdb 文件中描述的原子的名字；"c+o+n+ca"则是对应的"indentifier"，它表示我们要选择 pdb 文件中名字叫"ca+cb"的原子（ca 代表 alpha carbon，cb 代表 beta carbon）。整个语句的作用就是选择上诉的原子并命名为"test"，这样我们可以在后面继续使用它。

下表列出了大多数的 selector：

Selector	**简写**	**Identifier 及例子**
symbol	e.	chemical-symbol-list 周期表中的元素符号 Pymol> select polar, symbol o+n

name	n.	atom-name-list pdb 文件中的原子名字 Pymol> select carbons, name ca+cb+cg+cd
resn	r.	residue-name-list 氨基酸的名字 Pymol> select aas, resn asp+glu+asn+gln
resi	i.	residue-identifier-list pdb 文件中基团的编号 Pymol> select mults10, resi 1+10+100 residue-identifier-range Pymol> select nterm, resi 1-10
alt	alt	alternate-conformation-identifier-list 一些单字母的列表，选择具有 2 种构型的氨基酸 Pymol> select altconf, alt a+b
chain	c.	chain-identifier-list 一些单字母或数字的列表 Pymol> select firstch, chain a
segi	s.	segment-identifier-list 一些字母（最多 4 位）的列表 Pymol> select ligand, segi lig
flag	f.	flag-nummer 一个整数（0 － 3 1） Pymol> select f1, flag 0
numeric_type	nt.	type-nummer 一个整数 Pymol> select type1, nt. 5
text_type	tt.	type-string 一些字母（最多 4 位）的列表 Pymol> select subset, tt. HA+HC

id	id	external-index-number 一个整数 Pymol> select idno, id 23
index	idx.	internal-index-number 一个整数 Pymol> select intid, index 23
ss	ss	secondary-structure-type 代表该类结构的单字母 Pymol> select allstrs, ss h+s+l+""

下表是另一些 Selector，有关比较的：

Selector	简写	Identifier 及例子
b	b	comparison-operator b-factor-value 一个实数，用来比较 b-factor Pymol> select fuzzy, b > 12
q	q	comparison-operator occupancy-value 一个实数，用来比较 occupancy Pymol> select lowcharges, q > 0.5
formal_charge	fc.	comparison-operator formal charge-value 一个整数，用来比较 formal charge Pymol> select doubles, fc. = -1
partial_charge	pc.	comparison-operator partial charge-value 一个实数，用来比较 partial charge Pymol> select hicharges, pc. > -1

另外有一些 Selector 是不需要 Identifier 的，它们被列在下表中：

Selector	简写	描述
all	*	所有当前被 Pymol 加载的原子
none	none	什么也不选
hydro	h.	所有当前被 Pymol 加载的氢原子
hetatm	het	所有从蛋白质数据库 HETATM 记录中加载的原子

visible	v.	所有在被“可见”的显示的对象中的原子
present	pr.	所有的具有定义坐标的原子

在 Identifier 中用到的原子以及氨基酸的命名规则可以在下面的网址中找到：

http://www.wwpdb.org/docs.html

在选择表达中，selector 还可以配合逻辑操作子（logical operator）使用，这样我们可以表达更加复杂的选择。这些操作子被列于下表中：

Operator	**简写**	**效果与例子**
not s1	! s1	选择原子但不包括 s1 中的 Pymol> select sidechains, ! bb
s1 and s2	s1 & s2	选择既在 s1 又在 s2 中的原子 Pymol> select far_bb, bb & farfrm_ten
s1 or s2	s1 | s2	选择 s1 或者 s2 中的原子（也就是包含全部的 s1 和 s2 原子） Pymol> select all_prot, bb | sidechain
s1 in s2	s1 in s2	选择 s1 中的那些原子，其 identifiers (name, resi, resn, chain, segi) 全部符合 s2 中对应的原子 Pymol> select same_atom, pept in prot
s1 like s2	s1 l. s2	选择 s1 中的那些原子，其 identifiers (name, resi)符合 s2 中对应的原子 Pymol> select similar_atom, pept like prot
s1 gap X	s1 gap X	选择那些原子，其 van der Waals 半径至少和 s1 的 van der Waals 半径相差 X Pymol> select farfrm_ten, resi 10 gap 5
s1 around X	s1 a. X	选择以 s1 中任何原子为中心，X 为半径，所包括的所有原子 Pymol> select near_ten, resi 10 around 5
s1 expand X	s1 e. X	选择以 s1 中任何原子为中心，X 为半径，然后把 s1 扩展至该新的范围所包含的所有原子 Pymol> select near_ten_x, near10 expand 3
s1 within X of s2	s1 w. X of s2	选择以 s2 为中心，X 为半径，并包含在 s1 中的原子 Pymol> select bbnearten, bb w. 4 of resi 10
byres s1	br. s1	把选择扩展到全部 residue

		Pymol> select complete_res, br. bbnear10
byobject s1	bo. s1	把选择扩展到全部 object Pymol> select near_obj, bo. near_res
neighbor s1	nbr. s1	选择直接和 s1 相连的原子 Pymol> select vicinos, nbr. resi 10

这些逻辑选择还可以组合使用。比如你想选择 chain b，但是不选择其中的 residue 88：

```
Pymol> select chain b and (not resi 88)
```

在使用多重逻辑选择时，为了让 Pymol 正确处理顺序，请使用括号，这样最里层括号里面的内容将会被最先处理，以此类推。

好了，目标选择就先说到这里。其实关于目标选择还有所谓的“宏”可以用，可以简化表达式，准备下次说说。

Pymol 学习笔记（五）：Pymol 的选择宏

上次具体讲了如何在 Pymol 中怎么用 selection-expression 选取目标，其实在某些情况下，还可以用 Pymol 提供的宏来选择操作目标。使用这个选择宏往往可以是一个原本很复杂的表达变得简单紧凑。

例如我们想选择 2vlo 这个 pdb 文件中的"chain a"中的第 100 个基团的 α 炭原子，如果用 selection-expression 来表达的话是这样：

```
Pymol> select chain a and resi 100 and ca
```

如果用宏的，可以这样：

```
Pymol> select a/100/ca
```

是不是觉得简单了很多。好了，下面就来详细讲讲这个宏吧。

因为这个宏是用来选择目标的，所以我称之为选择宏，它用斜杠"/"来定义 Identifier，并且它使用上次介绍过的逻辑操作子"and"。

一个完整的，按顺序的选择宏的表达如下：

```
/object-name/segi-identifier/chain-identifier/resi-identifier/name-identifier
```

之所以说选择宏是有顺序的，是因为 Pymol 就是靠顺序判断每个斜杠后面的东东都是什么东东。

如果再细分一下的话，其实这个选择宏有 2 种写法，一个是带开头的斜杠，另一个是不带开头斜杠。区别是：

如果不以斜杠开头，那么 Pymol 则认为你的表达式的最后一项就是选择宏的末尾的最后一项，也就是 name-identifier。例如：

```
Pymol> show lines, a/100/ca
Pymol> show lines, 100/ca
```

如过以斜杠开头，那么 Pymol 就认为你是从选择宏的表达式的顶端开始的，也就是从/object-name 开始的。例如：

```
Pymol> zoom /2vl0//a/100/ca
Pymol> zoom /2vl0//a/100
```

细心的读者肯定发现了上面的例子中有两道斜杠中间什么内容也没有，不会是写错了吧？当然不是，其实在这种情况下 Pymol 会默认选择这个两道斜杠中被省略的 Identifier 列表中的全部元素，也就是说被省略的部分被 Pymol 当作了一个通配符。例如上例中我要选择全部的"segment"，所以我就把它给省略不写了，呵呵，方便吧。

在举些例子来说明一下：

```
Pymol> color green, a/142/
```

斜杠后面的"name-identifier"被省略了，所以第 142 号基团的素有原子都会变成绿色。

```
Pymol> shwo cartoon, a//
```

a 斜杠后面的"resi-identifier"以及最后斜杠后面的"name-identifier"被省略了，所以整个 a 链将以 cartoon 的方式被显示。

```
Pymol> zoom /2vl0//b
```

2 个斜杠间的"segi-identifier"被省略，所以所有的 b 链将被放大。

最后总结一下，Pymol 的选择宏必须至少包含一个斜杠"/"，以此来和 Pymol 的"select-expression"区分；并且不能包含空格，因为 Pymol 是把宏作为一个词来读取的；还有就是其实 Pymol 在执行宏的时候首先是把它翻译成正常的"select-expression"，然后再执行的。

Pymol 学习笔记（六）：关于 cartoon

cartoon 经常被用来显示一个蛋白质的总体结构，看起来也很漂亮。这次就来说说它的具体用法。

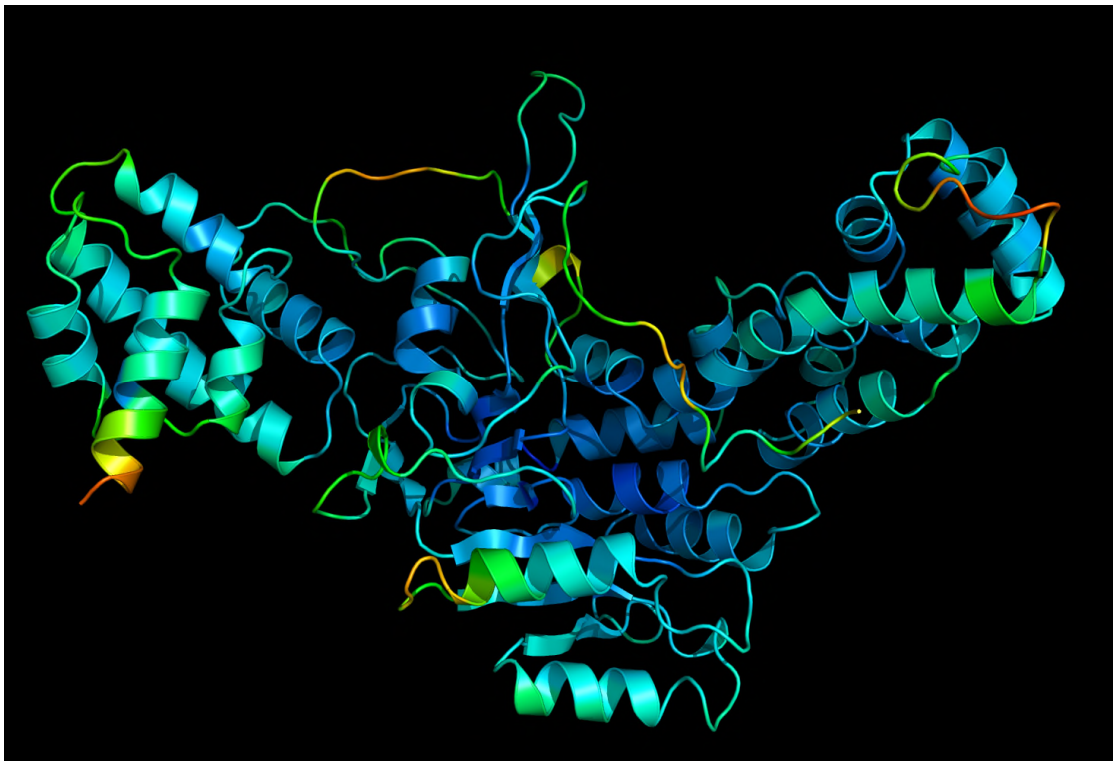
不久前本人刚搞定了一个 Glucosyltransferase 的结构，所以下面所有的例子都用来它来说明。

cartoon 的命令格式如下：

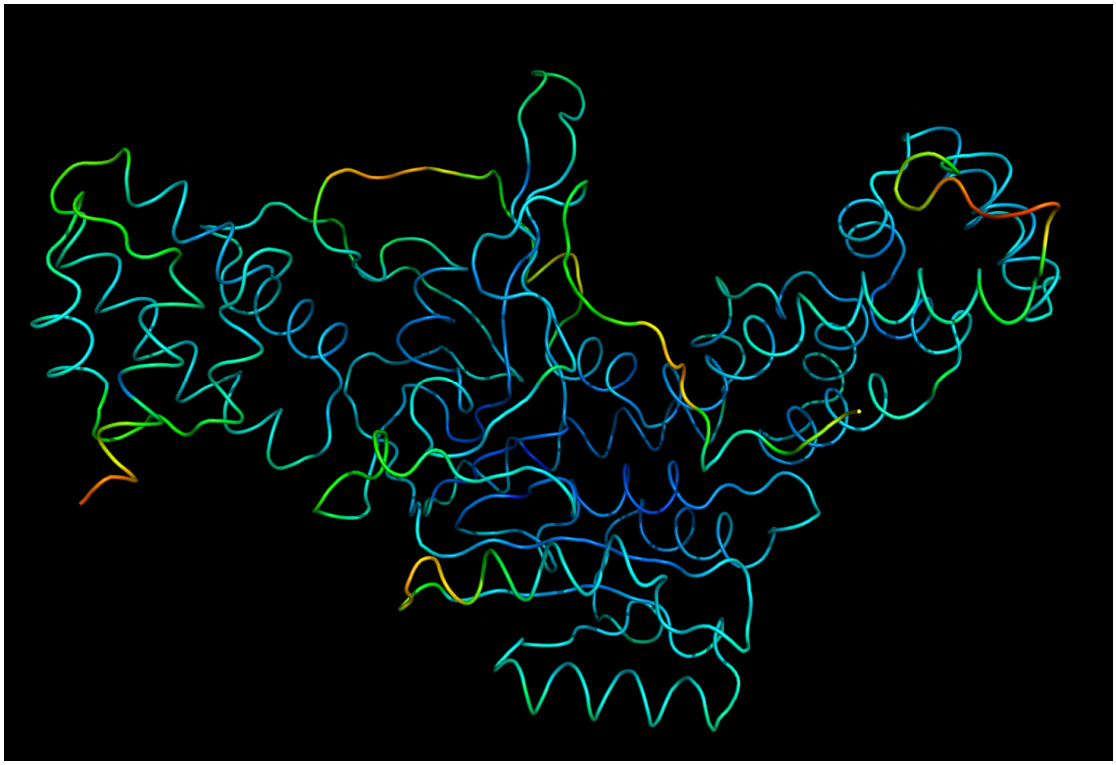
```
Pymol> cartoon type, (selection)
```

总结一下 cartoon 的显示类型：

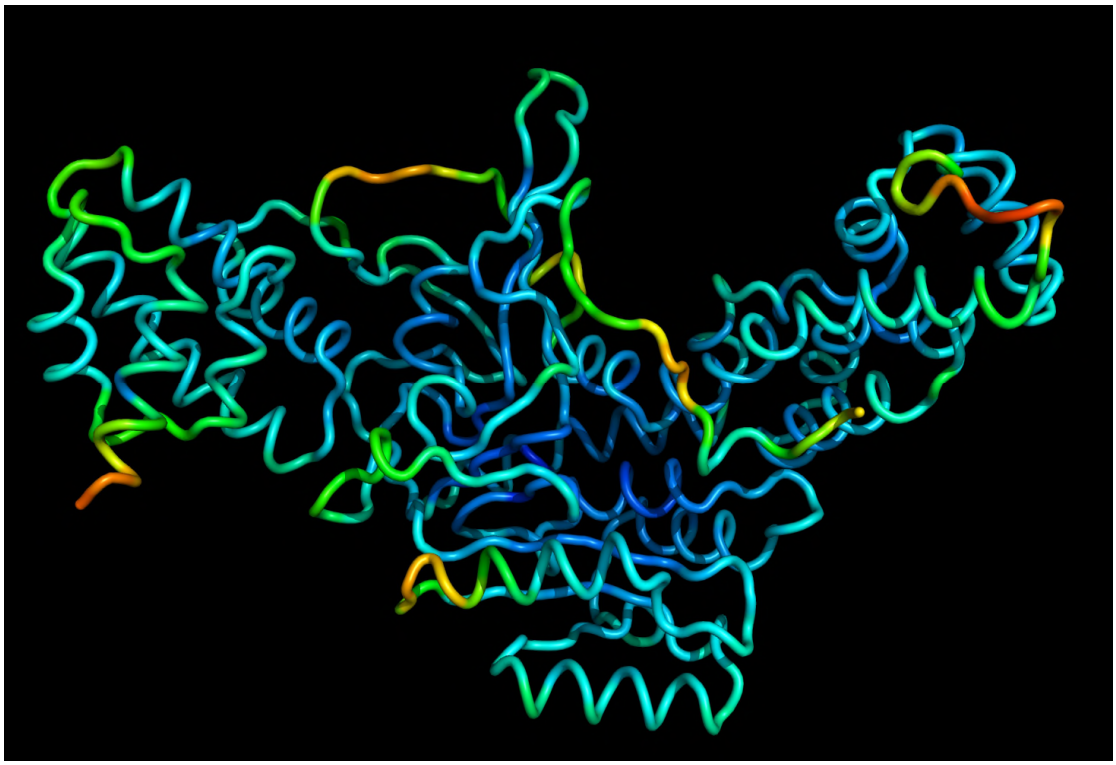
automatic：默认的显示方式



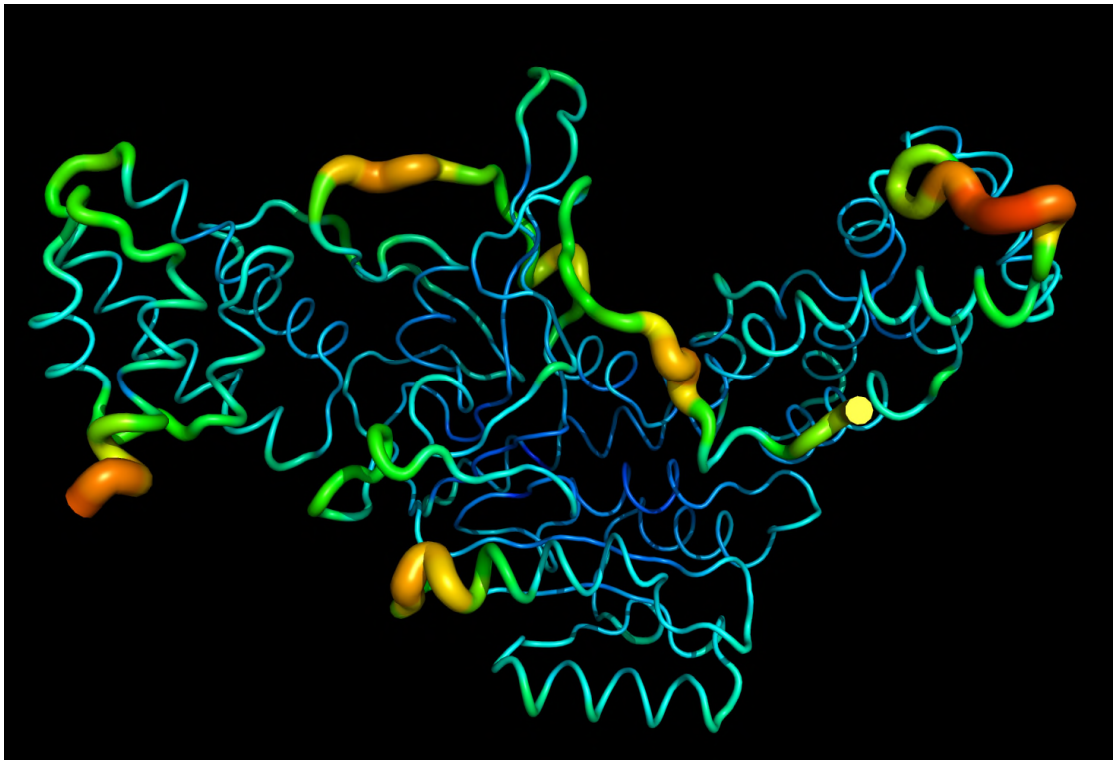
loop



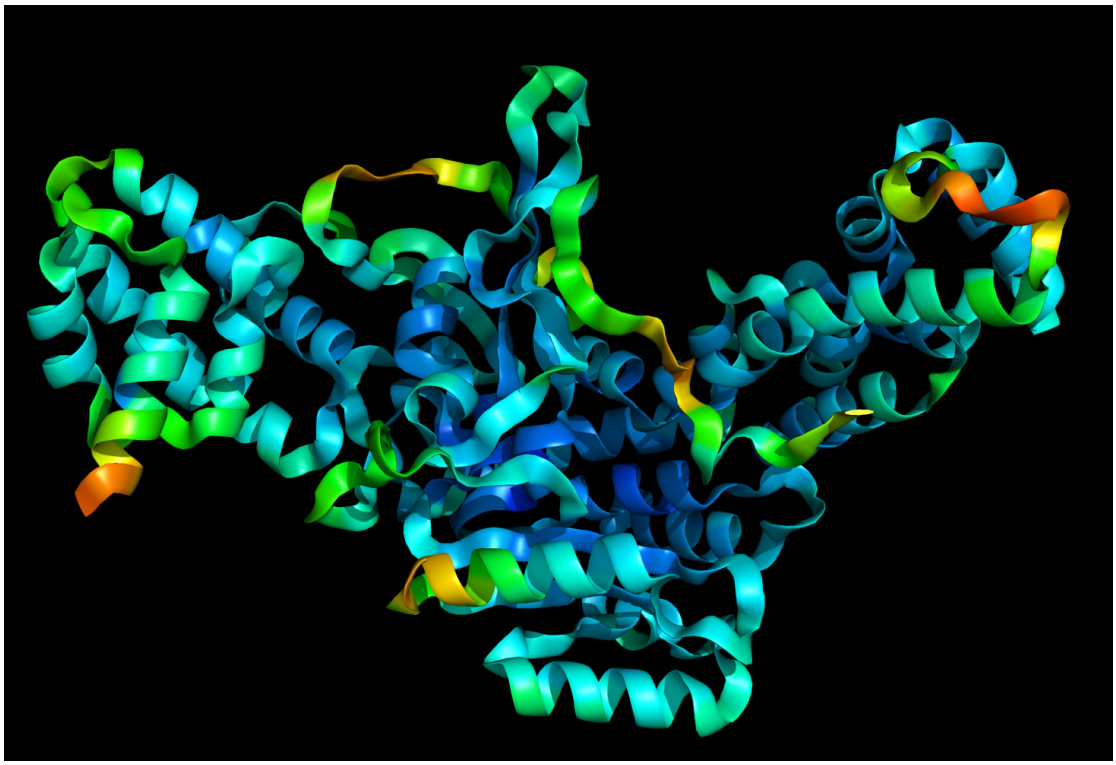
tube: 比 loop 粗点



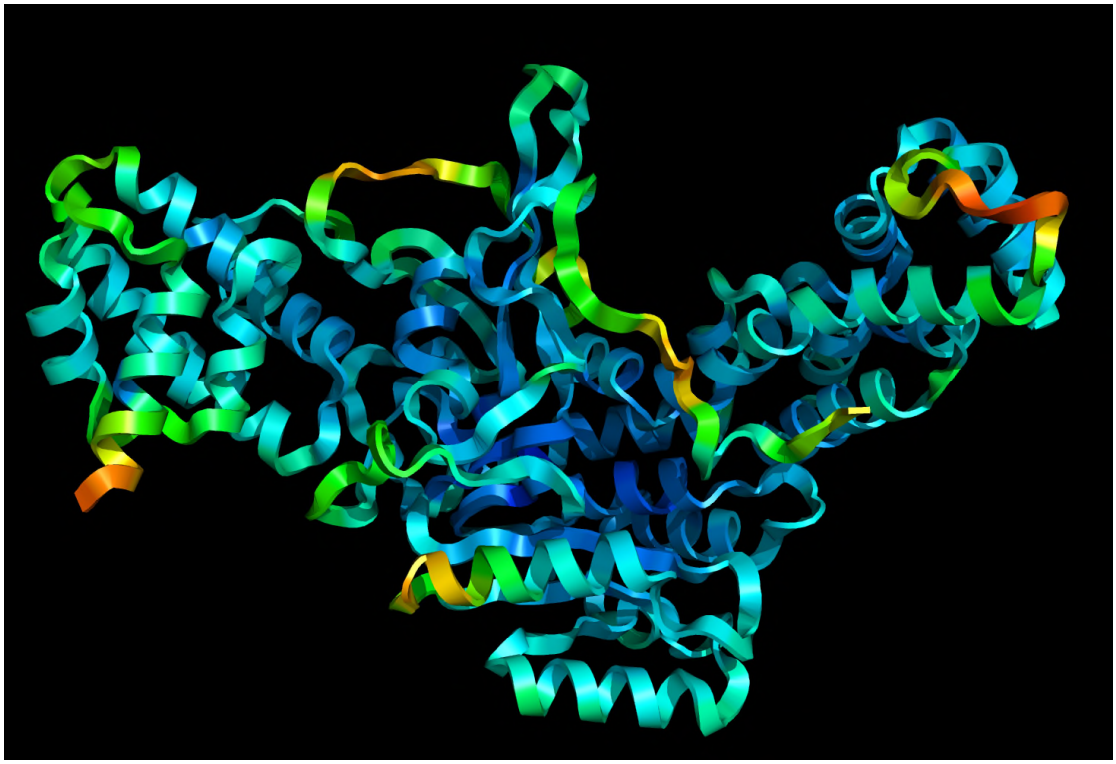
putty：这个比较有趣，按照 R-factor 来显示，越高越粗



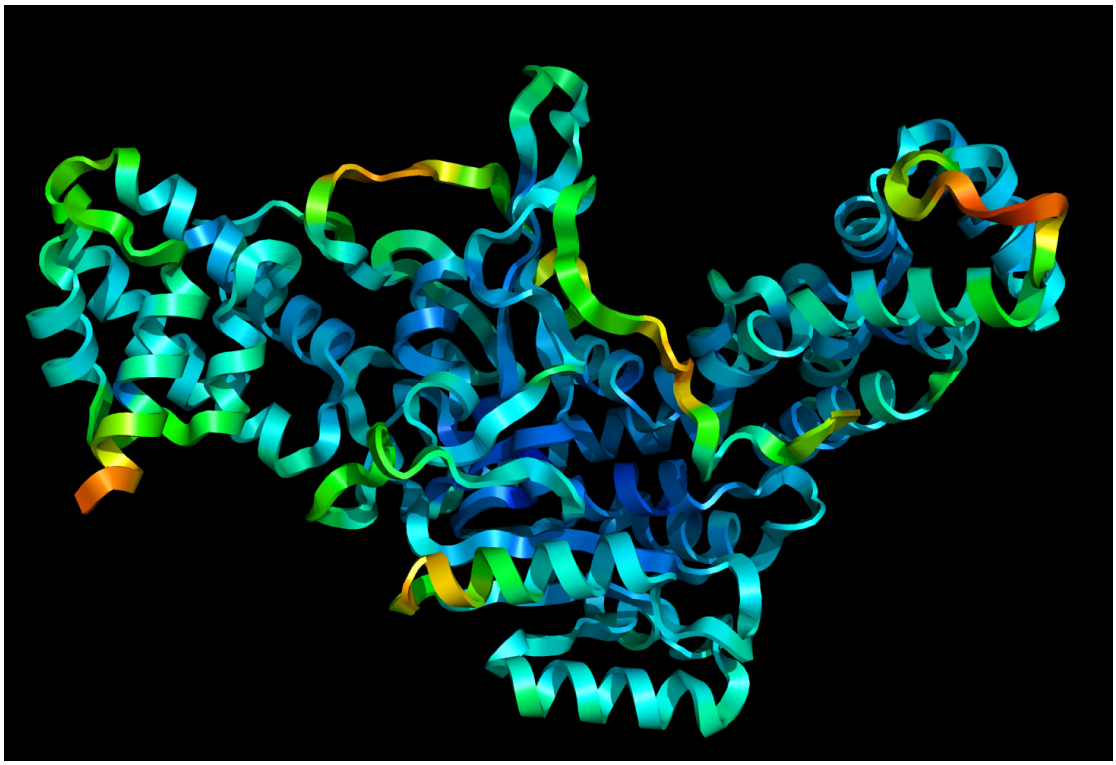
oval



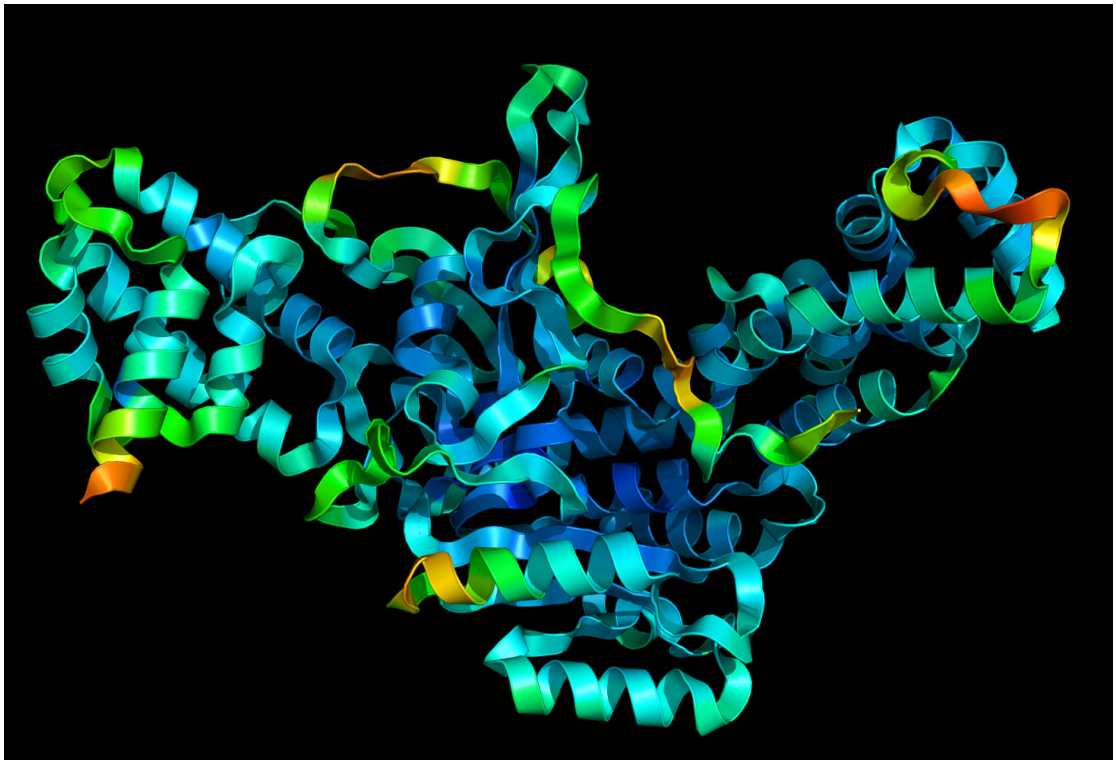
rectangle



arrow：和 rectangle 几乎一样，就是多了个箭头



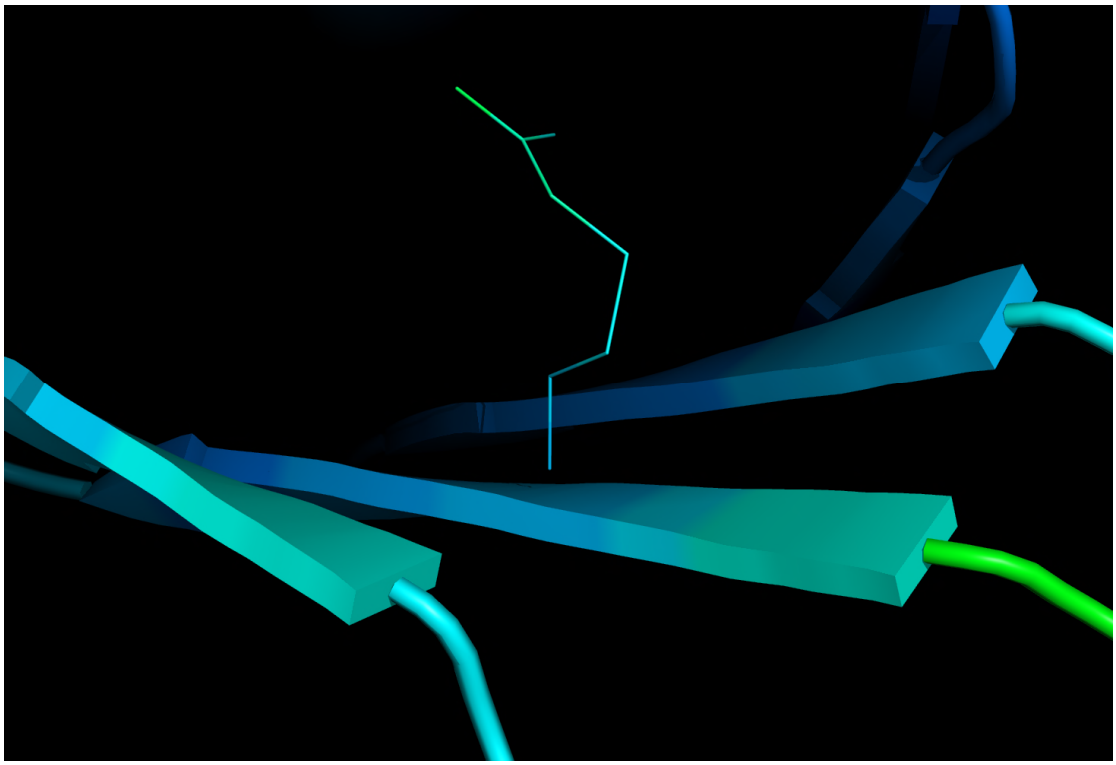
dumbbell：在 oval 的基础上，在 helix 的边缘加上一个 cylinder

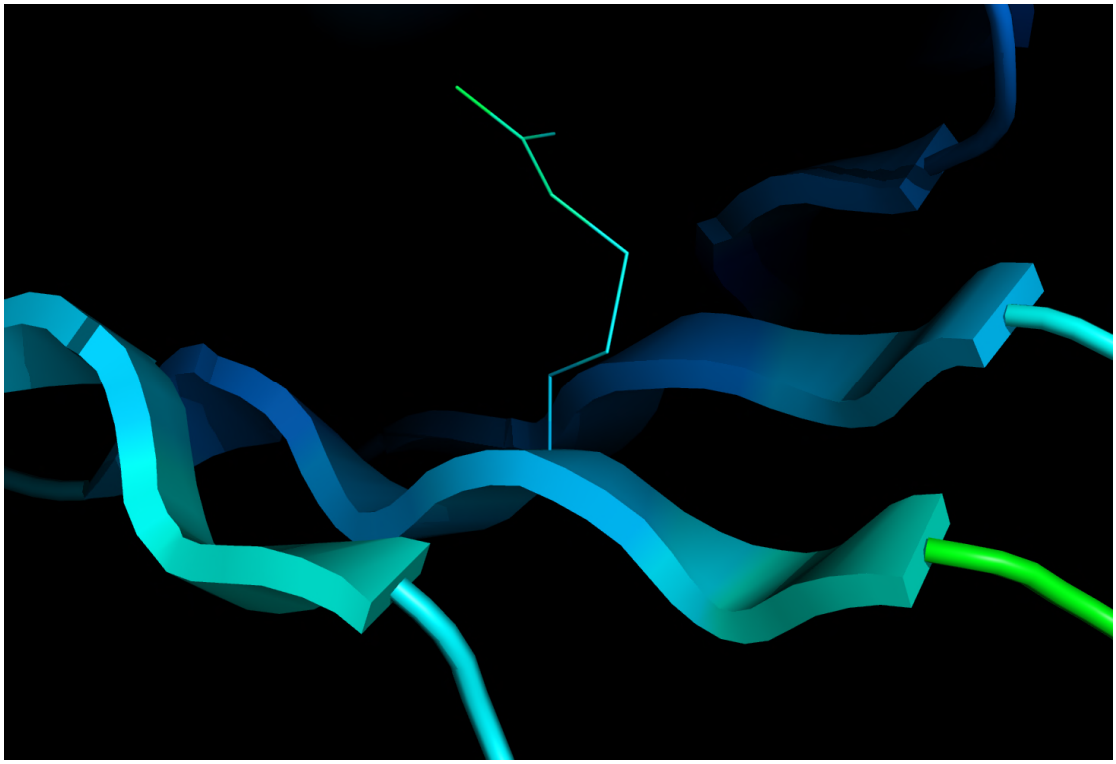


skip：隐藏，该图中隐藏了 6—120 号氨基酸。

下面说说如何设置 cartoon 的一些具体细节。

比较下面的 2 幅图：





你会发现第一张图中 sheets 是平的，而当中的那个氨基酸的支链并没连接在 sheet 上，这是因为为了显示的漂亮，把 sheet 人为的抹平了。而第二张图中的 sheets 则表达了蛋白质的真实走向，所以氨基酸的支链也显示正常。也就是说，如果你想表达某个局部的具体细节的时候，最好采用第二张图中的显示方式。2 张图对应的命令分别是：

```
Pymol> set cartoon_flat_sheets, 1
Pymol> set cartoon_flat_sheets, 0
```

类似的命令对应于 loop，就不举例子了：

```
Pymol> set cartoon_smooth_loops, 1
Pymol> set cartoon_smooth_loops, 0
```

下面再说说 cartoon 尺寸。

Helix 的厚度和宽度：

```
Pymol> set cartoon_oval_width, 0.2
Pymol> set cartoon_oval_length, 1.5
```

sheet 的厚度和宽度：

```
Pymol> set cartoon_rect_width, 0.5
Pymol> set cartoon_rect_length, 1.5
```

loop 的半径：

```
Pymol> set cartoon_loop_radius, 0.2
```

如果你设置了 cartoon 的显示风格为 fancy

```
Pymol> set cartoon_fancy_helices, 1
Pymol> set cartoon_fancy_sheets, 1
```

这样你得到的 helix 的边上会带有一个很细的 cylinder，也就是上面几张图中的显示方式。此时设置 helix 的厚度，宽度，以及这个 cylinder 的半径分别是：

```
Pymol> set cartoon_dumbbell_width, 0.1
Pymol> set cartoon_dumbbell_length, 2
Pymol> set cartoon_dumbbell, 0.2
```

依此类推，还可以设置和 putty，tube 等等显示类型相关的尺寸，就不一一类举了。

最后再加几个还用的着的命令吧：

上色：

```
Pymol> set cartoon_color, green
```

竟然还可以 refine，呵呵，逗号后面可以接数字，好像 1—20 都可以，数字越大优化的越大，感觉的确能变漂亮点：

```
Pymol> set cartoon_refine, 20
```

设置透明：

```
Pymol> set cartoon_transparency, 0.5
```

关于 cartoon 还有些命令，感觉不怎么常用，有些我也不知道是干什么的。有兴趣再研究吧。

Pymol 学习笔记（七）：关于 label

在显示一个蛋白结构的某个细节的时候，常常会需要给某些重要的氨基酸打上标签，这就需要用到 label 命令。

label 的命令格式如下：

```
Pymol> label [selection, [expression]]
```

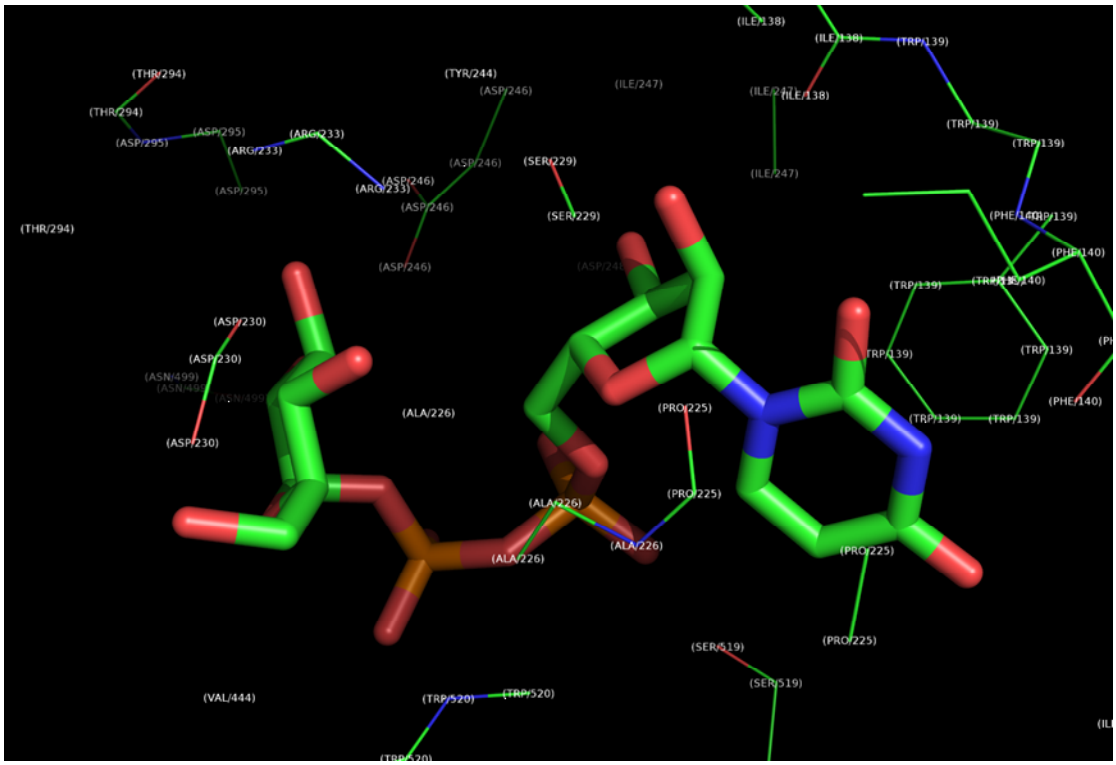
selection 当然就是你要加标签的对象，expression 就是标签的内容，可选的有：name, resn, resi, chain 等等。你也可以组合使用它们。expression 也可以是你自定义的一段内容，这时候只要把内容用引号包含起来就行：

```
Pymol> label selection, "user-defined expression"
```

在下面这个例子中，我想把 glucosyltransferase 中 UDP-Glucose 的 binding pocket 标注出来：

首先说明一下，该 pdb 文件中 A 链是蛋白质，B 链是 UDP-Glucose。

```
Pymol> load glucosyltransferase.pdb, tmp
Pymol> extract upg, chain b
Pymol> extract pro, chain a
Pymol> delete tmp
Pymol> select near, pro within 4.5 of upg
Pymol> hide all
Pymol> show sticks, upg
Pymol> show lines, near
Pymol> label near, ("%s/%s") % (resn, resi)#("%s/%s"):  设定显示格式。
```



上面的图看起来有点乱，因为默认 Pymol 在每个原子上都打上了标签。要想看起来顺眼点，需要一点加工。

在这之前，让我们先看一下关于 label 的其他设置：

投影模式，可选值 0（无投影），1（object 有投影到 label 上，但是 label 本身无投影），2（object 有投影到 label 上，label 也有投影），3（object 不投影到 label 上，label 本身有投影）：

```
Pymol> set label_shadow_mode, 3
```

文字颜色：

```
Pymol> set label_color, color-name, selection
```

标签文字的轮廓的颜色，这样就让在例如白色背景上加白色标签成为了可能：

```
Pymol> set label_outline_color, [color-name, [selection]]
```

字体，pymol 内置了 12 种字体，编号从 5—16。15 号和 16 号字体是 unicode 的：

```
Pymol> set label_font_id, 5
```

字体大小，如果为正值，则单位就是正常的 px。你也可以用负值，则单位是 Å：

```
Pymol> set label_size, -0.5
Pymol> set label_size, 4
```

设置 label 位置，用下列命令可以设置 label 离默认位置的三维偏移值，在需要给 spheres 加标签的时候有用：

```
Pymol> set label_position, (x,y,z)
```

最后说说怎样**用单个字母标注氨基酸**。

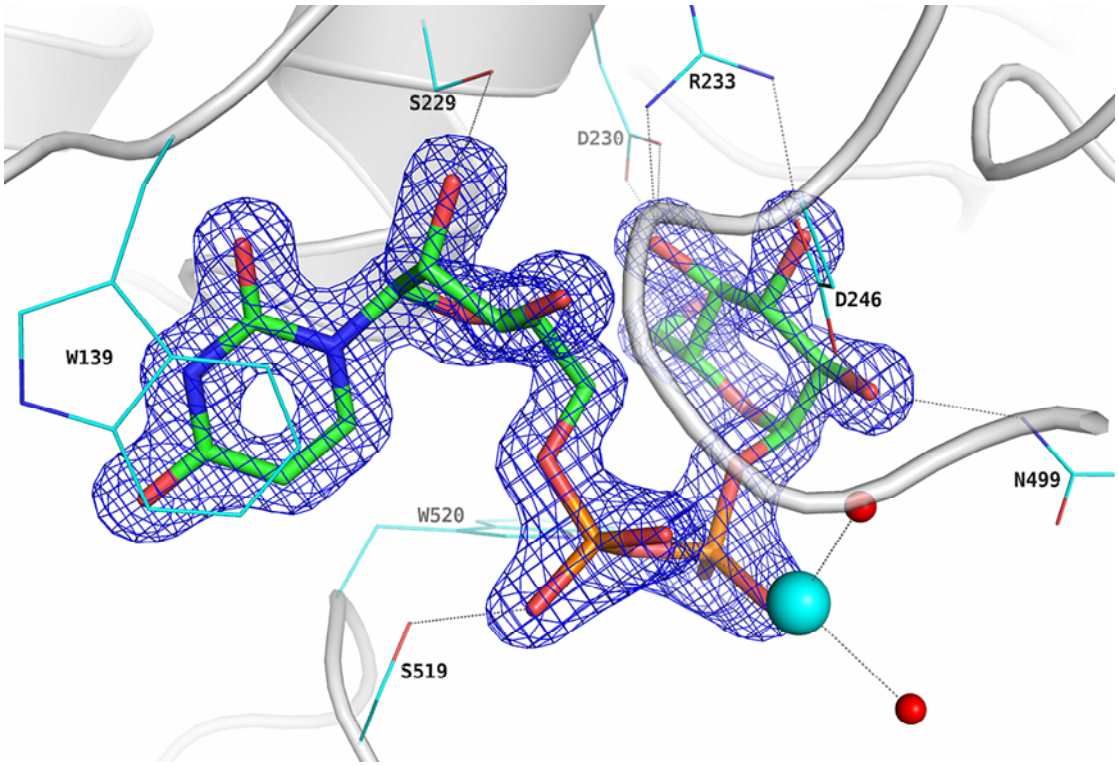
首先在$HOME/.pymolrc 中加入：

```
# start $HOME/.pymolrc modification
single ={'VAL':'V', 'ILE':'I', 'LEU':'L', 'GLU':'E', 'GLN':'Q', \
'ASP':'D', 'ASN':'N', 'HIS':'H', 'TRP':'W', 'PHE':'F', 'TYR':'Y', \
'ARG':'R', 'LYS':'K', 'SER':'S', 'THR':'T', 'MET':'M', 'ALA':'A', \
'GLY':'G', 'PRO':'P', 'CYS':'C'}
# end modification
```

用法，用 single[resn]代替 resn：

```
Pymol> label n. CG and i. 230+246, single[resn]
```

下面是改进过的图片：



是不是看起来好多了，下面是具体的代码，其中关于电子密度图的设置请见下一节：

```
Pymol> select near, resi 139+229+230+233+246+499+519+520
Pymol> as cartoon, pro
Pymol>  鼠标操作：显示 near 的 sidechain
Pymol> set_color grey1, [224,224,224]
Pymol> set cartoon_color, grey1
```

```
Pymol> set cartoon_transparency, 0.3
Pymol> set cartoon_fancy_helices, 1
Pymol> label n. CB and near, ("%s%s") % (single[resn], resi)
Pymol>  进入 Editing 模式，按住 ctrl+鼠标右键移动 label 到合适位置
Pymol> set cartoon_transparency, 0.3
Pymol> set label_font_id, 13
Pymol> set label_size, 26
Pymol> bg_color white

Pymol>  进入 measurement 模式测量我们所需要的距离
Pymol> set dash_length, 0.015
Pymol> set dash_radius, 0.015
Pymol> set dash_gap, 0.6
Pymol> set dash_color, grey
```

Pymol 学习笔记（八）：在 pymol 中导入电子密度图

假设我们已经有了一个蛋白质的 pdb 文件(protein.pdb)和 mtz(protein.mtz)文件，这样我们就可以用 fft(Fast Fourier Transform)命令生成 electron density map，以便在 pymol 中导入。

fft 的命令格式如下：

```
fft HKLIN protein.mtz XYZIN protein.pdb MAPOUT 2fofc.ccp4
```

回车后进入 fft 环境，还需要指定一些参数：

```
LABIN F1=FWT PHI=PHWT
GRID SAMPLE 5
END
```

上诉第 2 句“GRID...”用来指定生成电子密度图的精细程度。默认值是 3，表示生成的“网格”大小是最大分辨率的三分之一，以此类推。

回车之后电子密度图就搞定了，名字叫 2fofc.ccp4

然后就可以在 pymol 中导入了。首先导入 pdb 文件 glucosyltransferase.pdb，然后电子密度图：

首先说明一下，该 pdb 文件中 A 链是蛋白质，B 链是 UDP-Glucose。

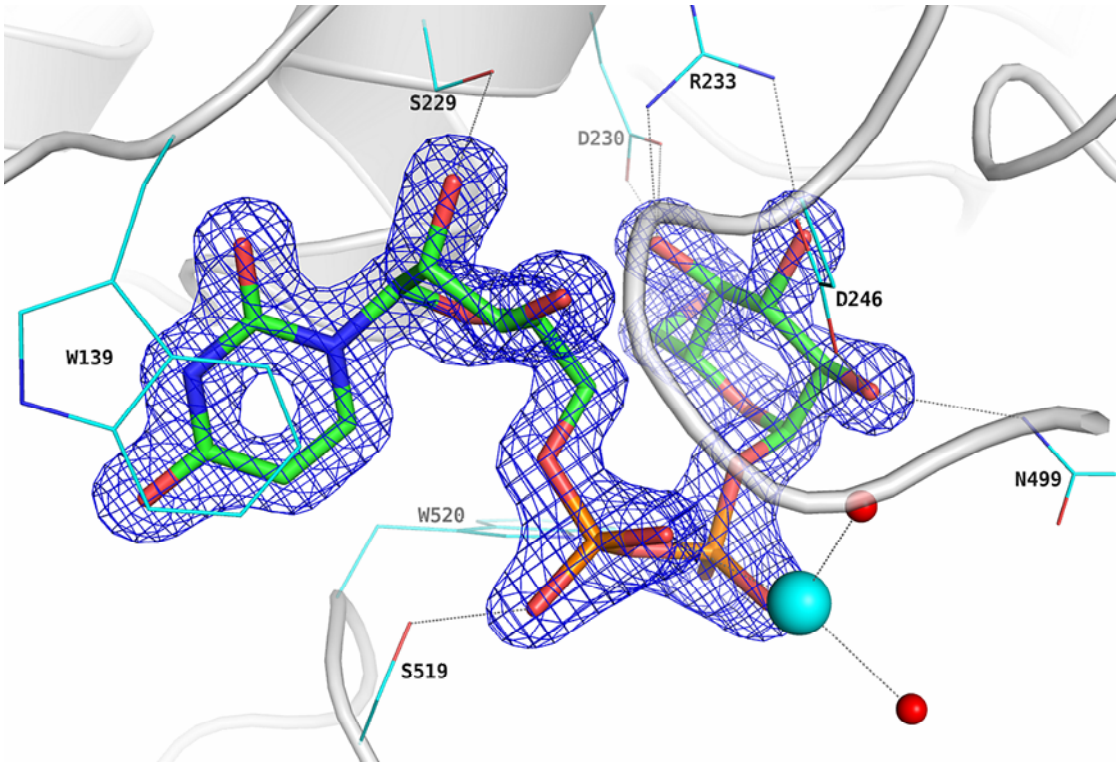
```
Pymol> load glucosyltransferase.pdb, pro
Pymol> load 2fofc.ccp4
```

你会发现，还看不见 density。打开 Wizard -- Density，怎么样，看见了吧。按住 ctrl 键和鼠标右键可以移动 density，或者点击 Density Map Wizard 中的"Next Res."和"Previous Res."。

不过显示全部的 density map 并不是我们的目的，因为这样显的很乱，也看不到什么细节。下面的例子是仅仅显示 UDP-Glucose 的电子密度图。

```
Pymol> remove resn HOH
Pymol> select upg, chain b
Pymol> as cartoon, pro
Pymol> as stick, upg
Pymol> orient, upg
Pymol> isomesh upg-d, 2fofc, 2.0, upg, carve=1.5
Pymol> set stick_radius, 0.2
Pymol> set mesh_radius, 0.01#让电子密度图的网格细一点，好看
```

效果图如下：



上诉命令中，其余设置请见上一节 label，和显示电子密度相关的就是 isomesh 了，它的用法如下：

```
Pymol> isomesh name, map, level [,(selection) [,buffer [,state [,carve ]]]]
```

- name: 给这个 mesh isosurface 随便起个名字。

- map: 就是刚才 load 的那个 2fofc。
- level: 轮廓值，越大轮廓越细。
- selection: 要表达的对象
- buffer: 没用过
- state: 没用过
- carve: 以 selection 中的任何原子为中心，半径为该值，所包含的全部原子，画出他们的 density

OK!搞定。