

## Unix & Common Unix Commands, Tips

Unix is a powerful text command based operating system, similar to DOS, but designed for a sophisticated multi-user environment. X-Windows provides a graphical user interface (GUI), but you need to learn to use some of the common Unix commands. What makes Unix so difficult to use is that most of us are used to PC-Windows or Macintosh operating systems that insulates us from the operating system with a GUI menu driven command structure, so we do not have to learn various command sequences to get the computer to do what we want.

Unix commands often have little resemblance to the abbreviations used by DOS that correspond to what you want to do. For example, if you want to get a listing of the files in your directory in Unix you use the **ls** command, NOT **dir**. This is another major source of confusion.

For additional info on Unix see:

<http://www.mhpcc.edu/training/vitecbids/UnixIntro/UnixIntro.html>

### Some Important General Unix notes:

- UNIX is **case sensitive**. It is important to remember that the files *IMAGE.JPG* and *image.jpg* are NOT the same. The normal convention in UNIX is to use *lower case* letters.
- In UNIX, you do not use the **backslash**, but rather the **forward slash** when referencing directories. So my home directory is: **/home/stanley/stanley**. But most of you may have root directories as such: **/usr/people/research-advisor/your-name**
- UNIX does not have an **undelete** command. Once you remove (delete) a file, it is gone forever. There is a chance that if the file has been around for more than 2 days that a backup of it to 4mm DAT tape has been made. It can then be restored.
- Command options in UNIX generally use the **hyphen-**, not the **forward slash** like DOS. Almost all Unix commands have many, many command options to do different specific sub-functions of the command.

## Change your Password

You should change your initial password very soon after your first login. To change your password: enter the command **passwd** and then respond to the prompts by entering your *old password* followed by your *new one*. You are then asked to retype your password for confirmation.

Note that what you type will not appear on the screen for security reasons. For example:

### **passwd**

Old password: *enter your current password*

New password: *enter your new password*

Retype new password: *re-enter your new password*

If you make a mistake, the message: **Mismatch - password unchanged.** is displayed and your password remains unchanged. Try again.

## Directories & Files

Unlike DOS, UNIX has no concept of drive letters. UNIX uses what are called mount points such as **/usr** or **/usr/local** or **/var**.

- **pwd** tells you what directory you are in.
- **cd** alone returns you to your **\$HOME** directory.
- **cd ..** takes you up one directory level.
- **cd *dirname*** moves you to the directory named "***dirname***".

## File Handling

Unix and DOS have the same utilities for dealing with files and directories - *only some of the names have been changed:*

- **ls** is the UNIX version of DOS **dir** (list files in directory)  
Syntax: **ls -l** shows long (detailed) listing.  
Syntax: **ls -last** shows long listing - newest first.
- **cp** Works just like *copy* in DOS.  
Syntax: **cp file1 file2**

**Warning:** *This can be quite tricky. See note below!!*

- **rm** Replaces the DOS *delete* command.  
Syntax: **rm filename**
- **mv** Does what *rename* does in DOS.  
Syntax: **mv file1 file2** (*potentially dangerous – be careful*)
- **mkdir** is the same as in DOS, creates a new directory.  
Syntax: **mkdir dirname**
- **rmdir** is the same as in DOS, deletes a directory.  
Syntax: **rmdir dirname**  
*\*\* all files and subdirectories must be removed first!!*
- **more** this is similar to *type* in DOS.  
Syntax: **more filename**  
This lists out the file one screen (page) at a time. Pressing the *Enter* key lists another “page” of the file. Pressing the “*space*” key lists out one line of the file at a time. *ONLY LIST OUT TEXT (ASCII) FILES. LISTING OUT A BINARY FILE WILL SCREW UP AND LOCK UP YOUR DISPLAY!!!*

The **copy (cp)** command, for those of you familiar with DOS, has some subtle differences that can make it difficult to initially use. Let’s say I’m copying a file called **sample.txt** from my current directory into a sub-directory called **exp1**. I would use the following Unix command:

```
cp sample.txt exp1
```

Note that I do NOT type in the name of the file in the target directory or use a \* (wild-card character) like in DOS. The above command will place a copy of **sample.txt** into the sub-directory **exp1**. If I want to change the name of the file, I do type in the new filename after the directory name as follows:

```
cp sample.txt exp1/sample2.txt
```

## File Permissions

DOS has a limited utility for setting permissions on files: *ATTRIB*. In UNIX has a more sophisticated method of setting permissions: the **chmod** command. To understand it, let's look at a small directory listing as made with the **ls -l** command.

Permissions	L	User	Group	Size	Creation Date	Filename
drwxrwxr-x	1	stanley	stanley	1018	Apr 30 23:45	catalysis.mdb
-rw-rw-r--	1	stanley	stanley	18755	Apr 30 23:37	image.gif
-rwxrwxr-x	1	stanley	stanley	18525	May 4 02:48	stan.tab

The file access permissions are indicated by a series of **rwX**'s on the left side of the listing. **The first position indicates the type of file.** For our purposes it is always a "-" for a file, or a **"d"** for a directory.

The remaining nine characters indicate 3 sets of permissions each given to the **owner**, **group**, and **others**, in that order. One can assign **read (r)**, **write (w)**, and **execute (x)** access permissions to each of these groups of users. You generally do not need to change these permissions. If you need help, see Cathie Griggs, the MCAF system manager.

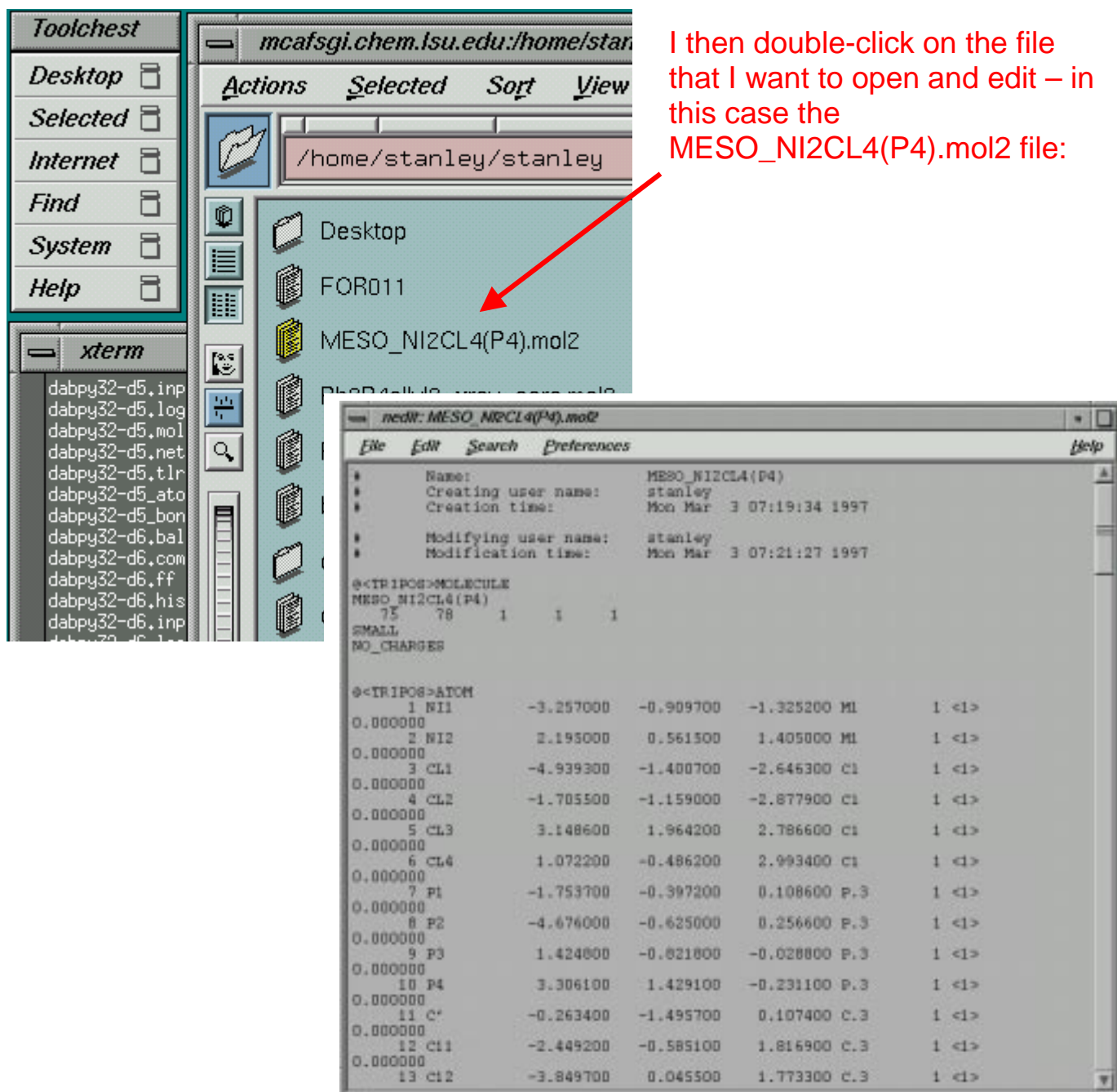
## Text Editors

The main editors available on most UNIX systems are:

- **vi**. The default text editor on almost all Unix systems. **Almost impossible to use** without practice prior to starting the program. It uses obscure commands to do everything and NONE of them, even on how to exit the program, are even vaguely obvious. DO NOT USE **vi** unless you have read up on it first. A good tutorial is located on the web at: <http://bignosebird.com/docs/vi.shtml>
- **emacs**. Powerful and easier to use than **vi** since it has a partial menu system to help you out and remind you of key text commands to do your editing. It is freeware and often installed on Unix systems (but not our SGI's).
- **pico**. On screen reminders of command keys, similar, but simpler than emacs. Not loaded on our SGI.

The text editor you want to use on the **SGI computers** is graphically based and easy to use (similar to a simple PC-Windows word processor). It is called **nedit** and can be started in one of two different ways. You can type in a command: **nedit filename** and a new X-window will open up with the contents of the file displayed ready for editing. Simply scroll up or down to find what you want to edit, click your mouse to where you want to edit, and type away. You can highlight text, cut and paste, etc.

The other way of starting **nedit**, is to have open the graphical window with your files open:



I then double-click on the file that I want to open and edit – in this case the MESO\_NI2CL4(P4).mol2 file:

```

@<TRIPOS>MOLECULE
MESO_NI2CL4(P4)
  75  78  1  1  1
SMALL
NO_CHARGES

@<TRIPOS>ATOM
  1 NI1      -3.257000  -0.909700  -1.325200  M1  1 <1>
0.000000
  2 NI2      2.195000   0.561500   1.405000  M1  1 <1>
0.000000
  3 CL1     -4.939300  -1.400700  -2.646300  C1  1 <1>
0.000000
  4 CL2     -1.705500  -1.159000  -2.877900  C1  1 <1>
0.000000
  5 CL3      3.148600   1.964200   2.786600  C1  1 <1>
0.000000
  6 CL4      1.072200  -0.486200   2.993400  C1  1 <1>
0.000000
  7 P1      -1.753700  -0.397200   0.108600  P.3  1 <1>
0.000000
  8 P2     -4.674000  -0.625000   0.256600  P.3  1 <1>
0.000000
  9 P3      1.424800  -0.021800  -0.028800  P.3  1 <1>
0.000000
 10 P4      3.306100   1.429100  -0.231100  P.3  1 <1>
0.000000
 11 C*     -0.263400  -1.495700   0.107400  C.3  1 <1>
0.000000
 12 C11    -2.449200  -0.585100   1.816900  C.3  1 <1>
0.000000
 13 C12    -3.849700   0.045500   1.773300  C.3  1 <1>

```

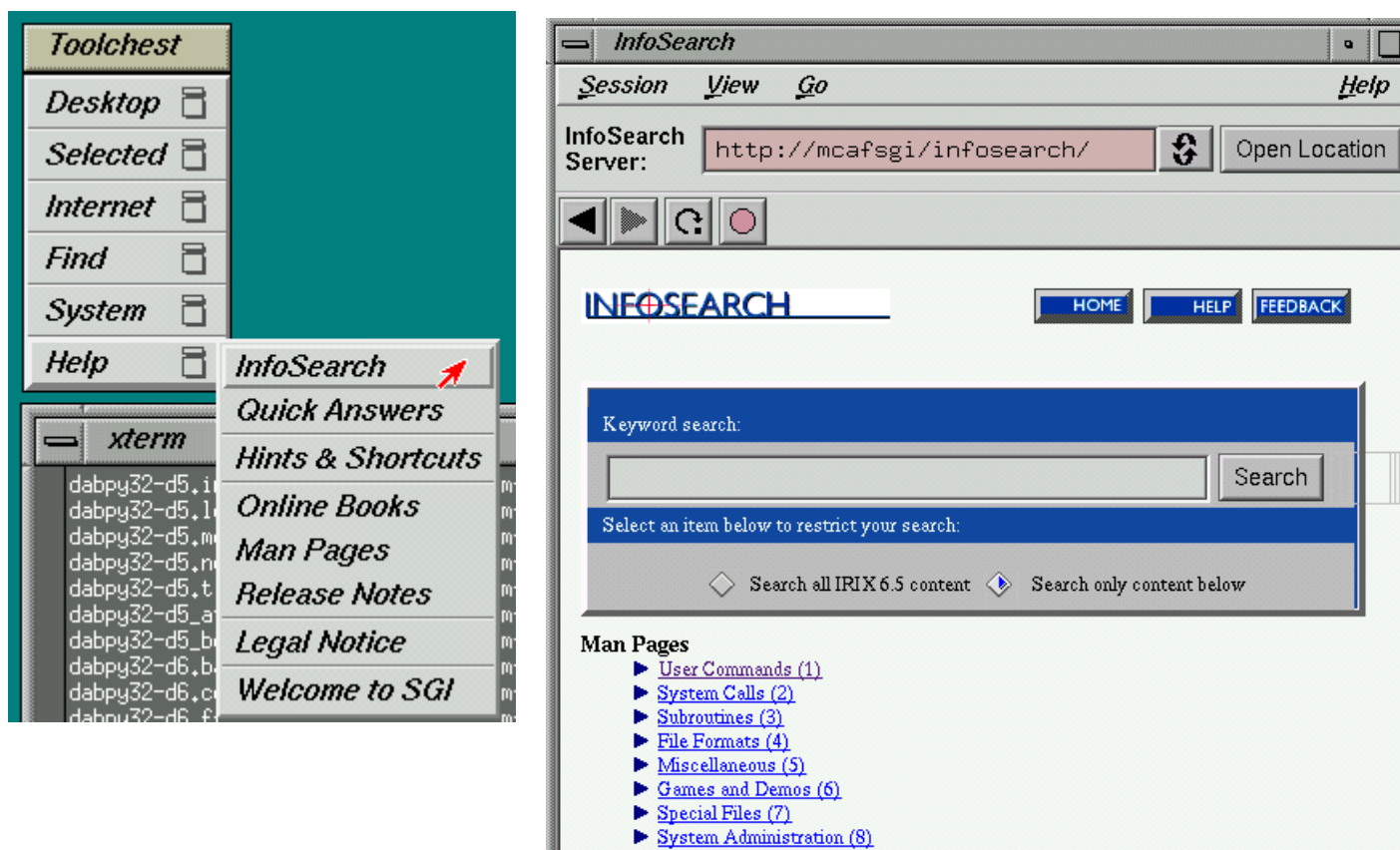
## HELP!

Every UNIX system has on-line documentation. It is identical to the paper manual pages. Getting help is as simple as entering the command:

### **man command**

The **man command** will format and display the manual pages for the command you request. Not all of the instructions you will find will make much sense as they were written for computer geeks, but many have examples and explanations that should give you some clue as to what is going on with that command.

The SGI has a nice GUI for accessing the **man** and other SGI specific help information. It is accessed from the **Toolchest Help** button:



For the above example, I then clicked on **man pages** to show what the **InfoSearch** window displays. Clicking on **User Commands** will bring up a big list of all the Unix commands and explanations of what they do and how to use them (unfortunately, in geek-talk).

The *Hints & Shortcuts* button under **Help** (above left) also is neat and I'll let you explore that on your own.

## FTP

You will be working on both the **mcafsgi** and **chonyx1** SGI computers in the MCAF. You will often need to transfer files between the two computer systems. We are working on implementing file sharing (the SGI Automount feature) between the two SGI computers, but until we have set it up and tested it, you will have to use **ftp** to transfer files between the SGI's or between the SGI's and your PC or Macintosh computers.

**ftp** stands for *File Transfer Protocol*. File transfer provides a means for you to obtain computer files (text, image, sound, etc.) from other computers over the network. **ftp** can also be used to send (upload) files from your computer to another computer, providing you have write permission or a real account on the machine you are uploading. The **ftp** utility has its own set of UNIX like commands that allow you to perform tasks such as:

- Connect and login to a remote host (**open computer-name**)

- Navigate directories (**cd**, just like Unix)

- List directory contents (**ls**, just like Unix)

- Put and get files (**put & get filename**)

- Transfer files as ascii, ebcdic or binary (use binary mode!!)

- Exit the ftp program (**quit**)

To **ftp** files between two computers, you need to have accounts on each of the computers. Some computers operate what is called *anonymous ftp*. You can login to these machines without a real account, to obtain files which have been made publicly available. Typically, the user name *anonymous* is used, coupled with your **email address as the password**.

Below is an example of copying a SYBYL directory (*protein.mdb*) from **chonyx1** to **mcafsgi** using **ftp**:

```
mcafsgi 5% mkdir protein.mdb
mcafsgi 6% cd protein.mdb
mcafsgi 7% ftp chonyx1.chem.lsu.edu
```

*Make a new directory called protein.mdb*

*Change directory into this new sub-directory we just created*

```
Connected to chonyx1.chem.lsu.edu.
220 chonyx1.chem.lsu.edu FTP server ready.
Name (chonyx1.chem.lsu.edu:stanley): stanley
331 Password required for stanley.
Password: typed in password
```

```
230 User stanley logged in.
```

```
Remote system type is UNIX.
```

Using binary mode to transfer files.

*Note that the SGI automatically put us into binary file transfer mode*

*Change directory to protein.mdb to access the files within that directory*

```
ftp> cd protein.mdb
```

```
250 CWD command successful.
```

```
ftp> mget *.*
```

```
mget INSULIN.mol2? y
```

```
200 PORT command successful.
```

```
150 Opening BINARY mode data connection for 'INSULIN.mol2' (115190 bytes).
```

```
226 Transfer complete.
```

```
115190 bytes received in 0.11 seconds (1045.07 Kbytes/s)
```

```
mget MONELLIN-b-sheetonly-Mclaughlin.mol2? y
```

```
200 PORT command successful.
```

```
150 Opening BINARY mode data connection for 'MONELLIN-b-sheetonly-Mclaughlin.mol2' (90218 bytes).
```

```
226 Transfer complete.
```

```
90218 bytes received in 0.09 seconds (991.09 Kbytes/s)
```

```
mget mark_dipeptide.mol2? y
```

```
200 PORT command successful.
```

```
150 Opening BINARY mode data connection for 'mark_dipeptide.mol2' (6398 bytes).
```

```
226 Transfer complete.
```

```
6398 bytes received in 0.01 seconds (977.02 Kbytes/s)
```

```
mget mark_dipeptide_stripped.mol2? n
```

```
mget mark_dipeptide_water_box.mol2? n
```

```
ftp> exit
```

*mget \*.\* is the multiple get command that will get all files present. It prompts you whether to copy each file (yes or no). To turn off the prompting feature, type in prompt BEFORE you mget*

```
?Invalid command
ftp> quit
221 Goodbye.
mcafsgi 8% ls
INSULIN.mol2          mark_dipeptide.mol2
MONELLIN-b-sheetonly-Mclaughlin.mol2
```

*The three Sybyl MOL2 files that we copied over from chonyx1*

## Who & What is Running on Unix?

Although Unix is a multi-user operating system, the computers will bog down if too many people are using the computer at the same time. Sybyl is a very cpu & memory intensive program and only about 2-3 people can be running Sybyl at the same time on the **Onyx** (the older SGI), while about double that # can run at the same time on the **Octane**.

As we get further along in the course and begin using Sybyl for more computationally intensive jobs, we will submit these to BATCH mode for execution. When you log onto Sybyl either at the SGI workstation or via X-windows, you are doing what is called interactive computing. You expect an immediate response from the computer when you issue a command or do something. Unix assigns a higher cpu priority to interactive jobs.

Batch mode is when a computationally intensive job is submitted to the "background." Once this is done you can Log off the computer and your job will keep running. Any Sybyl job that is going to take more than 10 mins of cpu time should be submitted to the batch queue. All Sybyl computations that take considerable cpu time have Batch mode options. Some can only be run via submission to the Batch queue.

When you log onto the SGI computer you should check how many other users are logged on. The **finger** command will give you this info. If too many users are on, logoff and wait till someone logs off (or try the other SGI). Note that if you are just playing with the Unix commands, that is not at all cpu time consuming. The SGI's can easily handle 10 users at a time for simple Unix command stuff (like file management).

## finger

The **finger** command displays information about users on a given computer. Some examples are shown below:

**finger** - show all logins on the current computer  
**finger @chonxy1.chem.lsu.edu** - show users on another computer system

output from finger:

Login	Name	TTY	Idle	When
rjw	Robert J West	p0	2:11	Sat 10:32
mdine	Marc Dine	p1	17:41	Wed 09:46
zepht	James S. Tallis	p2	7:11	Wed 09:46
bqs5	Barbara Baker	p3		Sat 10:33
davek	Dave Kiley	p4	8:42	Sat 10:33

**TTY** refers to the terminal port that Unix assigns to the person. **Idle** is the time that the person has sat doing nothing on the computer. **When** is when the person logged on.

## What Jobs are Running?

There are several ways to see what jobs are running on the system. None are perfect for telling you exactly who and what all they are running.

- 1) The Unix command **ps -af** will display all the interactive **user** jobs consuming computer time (no system, i.e., **root**, jobs are displayed, **nor are batch jobs shown**). The output looks like this:

```
mcafsgi 1% ps -af
UID      PID      PPID  C  STIME   TTY   TIME CMD
bbarker  282089   282098 0 01:48:46 pts/0  0:01 /usr/local/sybyl66/bin/sybyl.exe
bbarker  282143   282089 0 01:48:49 pts/0  0:00 /usr/local/sybyl66/bin/syb_heartbeat
stanley  282241   282141 0 01:49:18 pts/2  0:00 ps -af
```

User **bbarker** is running **Sybyl** (two processes associated with Sybyl are running, **sybyl.exe** is the main Sybyl program) while user **stanley** just ran the **ps -af** Unix command. **PID** is the process identification #, **STIME** is the starting time for the job (note that the system clock is currently way off), **TIME** is the cumulative cpu time for the job (hrs:mins), and **CMD** is the command that the computer is running with the full path to that command.

If you want to stop a job that is running, you issue the following command: **kill -9 pid#** where the *pid#* is the first PID (process identification # from the **ps -af** command). Note that you can only “kill” your own jobs (Cathie Griggs and George Stanley have **system privileges** and can “kill” anyone’s job if it gets “stuck”).

- 2) The Unix command **ps -aef |grep sybyl.exe** will display all the **Sybyl** jobs running. The vertical line character (|) immediately before the **grep** is the **shift \ key** above the **enter key** on your keyboard. Sybyl generates several processes for each job, but specifying **sybyl.exe** instead of just **sybyl** will narrow the list of jobs down to just the active executables. Any Sybyl job with a lot of cpu time is probably a batch job, while interactive Sybyl sessions usually don’t rack up that much cpu time. Using both the **ps -aef |grep sybyl.exe** and **ps -af** commands can allow you to identify the Sybyl batch jobs and interactive sessions.
- 3) If you are using X-windows (or on the SGI itself) you can issue the following command to open up a separate window that contains a list of active user jobs that is updated every couple of seconds: **gr\_top**

```

gr_top
IRIX64 mcafsgi 6.5 IP30      load averages: 1.66 0.70 0.27      02:09:11
72 processes: 69 sleeping, 1 ready, 2 running
2 CPUs: 0.0% idle, 77.8% usr, 7.6% ker, 0.0% wait, 0.0% xbrk, 14.6% intr
Memory: 1024M max, 952M avail, 694M free, 128M swap, 128M free swap

  PID   PGRP  USERNAME  PRI  SIZE  RES STATE  TIME  WCPU%  CPU%  COMMAND
  282540 595  stanley   18   62M   40M run/1   3:07  90.3  97.07 sybyl.e
  282692 595  stanley   18   62M   40M ready 1:10  88.9  95.25 sybyl.e
  282751 282751 stanley   20  2256K 1312K run/0   0:00   0.2   0.18 top
  282353 282353 stanley   20    38M   20M sleep 0:04   0.0   0.09 sybyl.e
  282389 282301 stanley   20  9312K 4704K sleep 0:00   0.1   0.04 4Dwm
  282733 282740 stanley   30  4592K 2528K sleep 0:00   0.0   0.04 xwsh
  282599 282596 stanley   30  4592K 2480K sleep 0:00   0.0   0.02 xwsh
    227 227  root    +118 2080K 1056K sleep 0:16   0.0   0.02 timesla
  282392 282301 stanley   20  4656K 2736K sleep 0:00   0.0   0.01 xterm
    744 210  root     20  2624K 1440K sleep 3:03   0.0   0.01 fam
   2386 2382  root     20  2144K 1424K sleep 0:33   0.0   0.01 lmgrd
    149 149  root     20  1728K  928K sleep 1:59   0.0   0.01 routed
    743 743  root     20  2816K 1632K sleep 5:49   0.0   0.01 mediad
    582 582  root     20  2704K 1840K sleep 0:12   0.0   0.00 sendmai
  
```

This gives detailed information on what jobs are actively consuming the cpu resources. **Do not leave **gr\_top** open for long since it consumes extra computer resources.** You can see from the second line that there are 72 processes, with 69 sleeping. The “sleeping” jobs are system jobs that perform specific functions every once in a while. They come and go

on this constantly updating display. The third line from the top shows that both cpu's are fully occupied (0.0% idle time). The fourth line shows that there is plenty of free memory available for jobs.

Below these top 4 lines is the list of jobs running. In a multi-tasking environment, the cpu (central processing unit) of a computer can usually only do one thing at a time. So when there are a number of jobs present demanding cpu time they are constantly swapped in and out, each grabbing a small amount of computer time. The accumulated computer time that the job has used is listed under the **TIME** column (usually listed as **min:sec**, but this will change to **hours** when you rack up enough computer time). The **CPU %** column shows how much of the two cpu's are working on those jobs. Since the Octane has two cpu's there is a total of 200% cpu time available. The **COMMAND** column gives you some indication of what job is running. There are **three** Sybyl jobs listed. I submitted two Sybyl molecular dynamic runs to the **batch queue** (the top two Sybyl jobs consuming the computer time) and I had an interactive Sybyl session running (the fourth line down with just 4 seconds of computer time used). Note that when you are running an interactive Sybyl session you are generally just sitting there looking at the screen and not using much cpu time.

The **SIZE** column refers to the amount of memory used by the various jobs. Each of the batch Sybyl jobs is using 64 MB of memory, while the interactive Sybyl session is only using 38 MB of memory (but this can jump up when you execute a command to actually do something).