Command-line Phylogenetics

A 'mini-workshop' in three parts developed by the Computational Evolutionary Biology (CEB) Group at the Florida State University School of Computational Science (SCS).

Part One:

Command-line Computing Basics

October 6, 2005, from 9:30 to 11:00 AM in Dirac 152

Author and Instructor: Steven M. Thompson

This portion of the workshop covers general background information on computers, the fundamentals of the UNIX/Linux operating system and the command line environment, and client/server computing connections and file transfers, as well as simple text editing.

Steve Thompson
BioInfo 4U
2538 Winnwood Circle
Valdosta, GA, USA 31601-7953
stevet@bio.fsu.edu
229-249-9751

Introduction

I write tutorials from a 'lowest-common-denominator' biologist's perspective. That is, I assume that you're relatively inexperienced regarding computers, especially command line computing. As a consequence of this my tutorials are written quite explicitly, and may even seem remedial. However, if you do exactly what is written, it will work. This requires two things: 1) you must read very carefully and not skim over vital steps, and 2) don't take offense if you already know what I'm discussing. I'm not trying to insult your intelligence.

I use three writing conventions. I use **bold** type for those commands and keystrokes that you are to type in at your keyboard or for buttons or menus that you are to click in a graphical user interface (GUI). I also use bold type for **section headings**. Screen traces are shown in a 'typewriter' style Courier font and "//////////" indicates abridged data. The 'greater-than' symbol (>) indicates the system prompt and should not be typed as a part of commands. Really important statements may be underlined.

To begin at the beginning, a computer is an electronic machine that performs rapid, complex calculations, and compiles and correlates data. It is composed of at least five basic parts: a central processor unit (CPU) that performs calculations, a data input device (such as a keyboard and mouse), a data output device (such as a monitor screen and printer), data storage devices (such as hard drives, floppy disks, and compact disks), and random access memory (RAM) where computing processes occur. Other necessary components include networking and graphics modules (boards) as well as the main architecture that it's all plugged into (the mother board). The quality, size, number, and speed of these components determine the type of computer: personal, workstation, server, mainframe, or super, though the terms are quite ambiguous.

Computers have a set of utility programs, called commands, known as an operating system (OS) that enable them to interact with human beings and other programs. OSs come in different 'flavors' with the major distinctions related to the company that originally developed the particular OS. Three primary OSs exist today with each having multitudes of variants: Microsoft (MS) Windows, Apple Macintosh OS, and UNIX. MS Windows, originally based on MS-DOS, is not UNIX at all. Apple's Mac OS, since OS X (version 10), is a true UNIX OS; earlier Mac OSs were not. UNIX OSs were originally proprietary, several are now Open Source. Server computers often use the RedHat Linux version of UNIX. RedHat is a commercial distribution of the free, UNIX derived, Open Source Linux OS. Linux was invented in the early 1990's by a student at the University of Helsinki in Finland named Linus Torvalds as a part-time 'hobby.' FreeBSD (from the U.C. Berkley UNIX implementation) is another popular Open Source UNIX OS. While all the various OSs have similar functions, the functions' names and their execution methods vary from one major class of OS to another. Most systems have a GUI to their OS providing mouse driven buttons and menus, and many provide a command line 'shell' interface as well.

The original UNIX OS was developed in the USA, first by Ken Thompson (no relation) and Dennis Ritchie at AT&T's BELL Labs in the late 1960's; it is now used in various implementations, on many different types of

computers the world over. All UNIX's are a line-oriented system similar conceptually to the old MS-DOS OS, though many GUIs exist to help drive them. It is possible to use many UNIX computers without ever learning command line mode. However, becoming familiar with some basic UNIX commands will make your computing experience much less frustrating. There's a great beginning UNIX tutorial at http://www.ee.surrey.ac.uk/Teaching/Unix/ if you would like to see an alternative to what I'll present here.

The UNIX command line is often regarded as very unfriendly compared to other OSs. Actually UNIX is quite straightforward, especially its file systems. UNIX is the precursor of most tree structured file systems including those used by MS-DOS, MS Windows, and the Macintosh OS. These file systems all consist of a tree of directories and subdirectories. The OS allows you to move about within and to manipulate this file system. A useful analogy is the file cabinet metaphor — your account is analogous to the entire file cabinet. Your directories are like the drawers of the cabinet, and subdirectories are like hanging folders of files within those drawers. Each hanging folder could have a number of manila folders within it, and so on, on down to individual files. Hopefully all arranged with some sort of logical organizational plan. Your computer account should be similarly arranged.

Computers are usually connected to other computers in a network, particularly in an academic or industrial setting. These networks consist of computers, switching devices, and a high-speed combination of copper and fiber optic cabling. Sometimes many computers are networked together into a configuration known as a cluster, where computing power can be spread across the individual members of the cluster (nodes). An extreme example of this is called grid computing where the nodes may be spread all over the world. Individual computers are most often networked to larger computers called servers as well as to each other. The worldwide system of interconnected, networked computers is called the Internet. Various software programs enable computers to communicate with one another across the Internet.

Most all computers have some type of a graphics-based browser, such as Explorer, Navigator, Firefox, Safari, Konqueror, Opera, on *ad infinitum*, it doesn't matter, they all access the World Wide Web (WWW), one part of the Internet, but only one part. You can use whatever browser is available to connect to WWW sites, identified by their Uniform Resource Locator (URL). The computer's Internet name is that portion of the URL just after the http:// and before the next slash (/). Unfortunately a Web browser alone is not enough, and won't do you any good in running the phylogenetic inference programs this workshop deals with. You need to directly connect to server computers using a command line, "terminal," window where you can directly interact with the server computer's OS. The 'old way' to do this was with a program named telnet. However, telnet is an insecure program from which smart hackers can 'sniff' connected account names and passwords. Therefore, in this age of the hacker, most server computers no longer allow telnet connections. A newer program named ssh, for 'secure shell,' encrypts all connections, and is now required for command line access to most servers. ssh comes preinstalled as a part of all modern UNIX OSs, but doesn't come with pre OS X Macs or any MS Windows machines and, therefore, must be installed on those platforms separately. The programs Nifty Telnet-SSH (http://www.msi.umn.edu/user_support/ssh/nifty_os9.html) and Putty

(http://www.chiark.greenend.org.uk/~sgtatham/putty/) are two free, public-domain ssh clients available for those respective platforms. ssh is already installed on all public use Biology Macs and MS Windows boxes and on all SCS machines.

Along the lines of secure connections, there are often times when you'll need to move files back and forth between your own computer and a server computer located somewhere else. The 'old' insecure way of doing this was a program named ftp, for file transfer protocol. Just like telnet it has the unfortunate attribute of allowing hackers to 'sniff' account names and passwords. Therefore, an encrypted file transfer counterpart to ssh is now required by most servers. That counterpart has two forms, sftp and scp, for 'secure file transfer protocol' and 'secure copy' respectively. It's also included in all modern UNIX OSs but not in pre OS X Macs nor in MS Windows. These programs are also installed on all the Biology and SCS common use computers.

Command line phylogenetics at Florida State University — a tutorial

The School of Computational Science (SCS) computer classroom has UNIX (Linux) workstations. Biology Department computer laboratories are equipped with Macintosh and MS Windows personal computers. Most servers at both sites are UNIX, consisting of a mixture of SGI/IRIX, Sun/Solaris, x86/Linux, x86/FreeBSD, and PPC/OS X. Any computer can be used to access these servers, as long as you use a legitimate account and the communication programs described above. Connecting from your home, apartment, dorm room, or anywhere in the world with Internet access, is entirely possible, as long as you've got these programs.

Let's begin with some general information from the URLs http://bio.fsu.edu/, http://www.bio.fsu.edu/, and http://www.csit.fsu.edu/. Activate the SCS classroom computer that you are sitting at by moving its mouse or by pressing the return/enter key on its keyboard. Log onto it with your new SCS account user ID and password. You'll see a standard SCS Linux desktop. Launch whatever WWW browser is available — find the icon on the desktop or in the GUI menu system. When the browser has opened, first go to the Web page for this workshop, **https://www.csit.fsu.edu/twiki/bin/view/TechHelp/MiniWorkshop**, where you can download each session's tutorial in Adobe's Portable Document Format (pdf, a device independent page description language; get Adobe's free pdf Reader, if you don't already have it). This Web page is built in a collaborative Web environment called TWiki that allows interactive editing by all of the instructors. Next visit the home page for the School of Computational Science (SCS) at **http://www.csit.fsu.edu/**. SCS provides the classroom and server computing environment that we are using for the workshop.

Now go to my home page, **http://bio.fsu.edu/~stevet/cv.html**, and look it over. Notice the links to the different lab tutorials I've developed. Briefly check out my biocomputing "**Bookmark**" list. Next, explore the navigation links in the sidebar. Use the "**Biology Department**" sidebar link to go to the main FSU Biology Department home page, **http://www.bio.fsu.edu/**. Among all the helpful links in its sidebar, pay particular attention to "**Research Facilities**;" press the button. Next, select "**Computer Support Facilities**" and then "**Labs**," to see the all the Biology computer labs. Go back to the main "Computer Facilities" page and select "**Help**" there. Under "**Internet and Email**" select "**How to connect to Mendel (mendel.csit.fsu.edu)**." Alex

Stuy, the Biology Department's senior computer support person, has written excellent instructions there for using ssh to connect to one of the SCS special use biocomputing servers, Mendel, that can be extended to any UNIX server.

So how do people do computational phylogenetics? Sometimes the WWW is used, especially in early data acquisition and alignment steps. This is possible and easy and fun, but, beside being a bit too easy too get sidetracked upon, the Web can't readily handle large multiple sequence alignment datasets. They quickly become intractable. You'll know you're there when you try. What are alternatives to the Web? Desktop software can be installed on your own personal computer. Free, public domain biocomputing programs are available, but they can be somewhat complicated to install, configure, and maintain. The user must be pretty computer savvy, especially to get them to all cooperate with one another. Therefore, commercial packages, such as MacVector, DNAsis, DNAStar, etc., can be used, but license hassles, a big expense per machine, and database access all complicate matters. Server based technologies can help. I manage a comprehensive commercial sequence analysis package (GCG) on Mendel for these types of chores.

In particular many biologists are familiar with the phylogenetic inference software PAUP* in the Mac 'Classic' GUI environment. However, as you've undoubtedly seen, and probably one of the reasons that you are in this workshop, running PAUP* in GUI mode on a personal Mac computer is not incredibly efficient, as any other running applications dramatically slow down its progress. Using a server-based PAUP* application can take the load off your personal computer, to free it up for more important things, like that latest manuscript you're trying to write. Moreover, the UNIX server environment is inherently more stable than your desktop environment — this helps keep your multiple day analyses running for as long as it takes. Furthermore, other phylogenetic inference software, such as MrBayes, is also used through a command line interface, and often on a server. Getting access to these resources requires network access to UNIX server computers, and the ability to comfortably operate in the UNIX command line environment.

Basic UNIX command line operation

SCS biocomputing servers only allow ssh, scp, and sftp connections. The same SCS computing account used to login to the classroom workstations will allow you to use the SCS general use servers. On any UNIX system (like the SCS classroom Linux workstations), launch a terminal program window with the appropriate icon from the desktop or from one of the menus ("**terminal**" from "**System Tools**" on most RedHat menus). Issue the command "**`ssh user@phoenix.csit.fsu.edu`**" (not typing the quotes and replacing "**`user`**" with your SCS account name) in the new terminal window to connect to the SCS general use phoenix cluster:

```
> ssh user@phoenix.csit.fsu.edu
```

If this is the first time that you've connected to phoenix from your local machine, you'll get something similar to the following screen trace, asking if you really want to do what you're doing. Answer the question with "**yes**" spelled all the way out:

```
The authenticity of host 'phoenix.csit.fsu.edu (144.174.160.169)' can't be
established.
RSA key fingerprint is 99:6d:e4:a3:0e:16:23:27:eb:31:1f:3d:91:af:8a:cc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'phoenix.csit.fsu.edu' (RSA) to the list of known hosts.
```

Next you'll be asked for your SCS account password. Note that passwords are not displayed in terminal windows as you type them. The screen trace continues after you've entered your password:

```
thompson@phoenix.csit.fsu.edu's password:

Welcome to the Phoenix Cluster.

Condor is available for job submission: /usr/local/condor.
In the future, I will have CSIT specific information on our local website.
Until then, point your web browser to http://www.cs.wisc.edu/condor for more
information.

If you are using the MPI universe with condor, mpich-1.2.4 is available in
/usr/local/mpich-1.2.4/ch_p4.
Direct submission of jobs is not allowed at this time. Please, utilize condor instead
so that everyone has an equal opportunity.

Please, subscribe to the condor-csit mailing list for cluster news:
http://lists.csit.fsu.edu/cgi-bin/mailman/listinfo/condor-csit

Thanks,

Tim Nguyen
tqnguyen@csit.fsu.edu
```

On pre-OS X Macs or MS Windows machines find and use the appropriate icon to launch their respective ssh GUI, either on the desktop or in menus. After a GUI ssh program opens, use the appropriate menu command to connect to phoenix.csit.fsu.edu. Regardless of the ssh method used to connect to phoenix, you should now have an interactive command line terminal session 'talking' to phoenix running on your local machine's desktop. Phoenix's OS checked your username and password, and if correct, ran your default shell program and any startup scripts, and then returned the system prompt. The shell program is your interface to the UNIX OS. It interprets and executes the commands that you type. Common UNIX shells include bash, the C shell, and a popular C shell derivative called tcsh. tcsh, like bash, enables command history recall using the keyboard arrow keys, accepts tab word completion, and allows command line editing.

Upon logging in, you end up in your 'home directory,' that portion of SCS computing disk space reserved just for you and designated by you from anywhere on the system with the character string "`$HOME`". This is known as an "environment variable." The system prompt displays the user and the machine name and waits to receive a command. On different UNIX systems the prompt may appear different ways depending on how the system administrator has set up the user environment. Here I will use the 'greater than' sign (>) to represent the system prompt. It should not be typed as part of the command.

In command line mode each command is terminated by the 'return' or 'enter' key. UNIX uses the ASCII character set and unlike some OSs, it supports both upper and lower case. A disadvantage of using both upper and lower case is that commands and file names must be typed in the correct case. Most UNIX

commands and file names are in lower case. Commands and file names should not include spaces nor any punctuation other than periods (.), hyphens (–), or underscores (_). UNIX command options are specified by a required space and the hyphen character (-). UNIX does not use or directly support function keys. Special functions are generally invoked using the 'Control' key. For example a running command can be aborted by pressing the "Control" key [sometimes labeled "CTRL" or denoted with the karat symbol (^)] and the letter key "c" (think c for cancel). The short form for this is generally written CTRL-C or ^C. Using control keys instead of special function keys for special commands can be hard to remember, the advantage is that nearly every terminal program supports the control key, allowing UNIX to be used from a wide variety of different platforms that might connect to the server.

The general command syntax for UNIX is a command followed by some options, and then some parameters. If a command reads input, the default input for the command will often come from the interactive terminal window. The output from a system level command (if any) will generally be printed back to your terminal window. General UNIX command syntax follows:

```
cmd
cmd -options
cmd -options parameters
```

The command syntax allows the input and outputs for a program to be redirected into files. To cause a command to read from a file rather than from the terminal, the "<" sign is used on the command line, and the ">" sign causes the program to write its output to a file (for programs that don't do this by default, also ">>" appends output to the end of an existing file):

```
cmd -options parameters < input
cmd -options parameters > output
cmd -options parameters < input > output
```

To cause the output from one program to be passed to another program as input a vertical bar (|), known as the "pipe," is used. This character is < shift > < \ > on most USA keyboards:

```
cmd1 -options parameters | cmd2 –options parameters
```

This feature is called "piping" the output of one program into the input of another.

Certain printing (non-control) characters, called "shell metacharacters," have special meanings to the UNIX shell. You rarely type shell metacharacters on the command line because they are punctuation characters. However, if you need to specify a filename accidentally containing one, turn off its special meaning by preceding the metacharacter with a "\" (backslash) character or enclose the filename in "'" (single quotes). The metacharacters "*" (asterisk), "?" (question mark), and "~" (tilde) are used for the shell file name "globbing" facility. When the shell encounters a command line word with a leading "~", or with "*" or "?"

anywhere on the command line, it attempts to expand that word to a list of matching file names using the following rules: A leading "~" expands to the home directory of a particular user. Each "*" is interpreted as a specification for zero or more of any character. Each "?" is interpreted as a specification for exactly one of any character, i.e.:

~	The tilde specifies the user's home directory (C shell and tcsh only, same as $HOME).
*	The asterisk matches any string of characters zero or longer,
?	The question mark matches any single character.

The latter two globbing shell metacharacters cause 'wild card expansion.' For example, the pattern "dog*" will access any file that begins with the word dog, regardless of what follows. It will find matches for, among others, files named "dog," "'doggone," and "doggy." The pattern "d?g" matches dog, dig, and dug but not ding, dang, or dogs; "dog?" finds files named "dogs" but not "dog" or "doggy." Using an asterisk or question mark in this manner is called using a "wild card." Generally when a UNIX command expects a file name, "cmd filename," it's possible to specify a group of files using a wild card expression.

A couple of examples using wild card characters along with the pipe and output redirection follow:

```
cmd */*.data | cmd2
cmd my.data? > filename
```

The first example will access all files ending in ".data" in all subdirectories one level below the current directory and pass that output on to the second command. The second example will access all files named "my.data" that have any single character after the word data in your current directory and output that result to a file named filename. Wild cards are very flexible in UNIX and this makes them very powerful, but you must be extremely careful when using them with destructive commands like "rm" (remove file).

Four other special symbols should be described before going on to specific UNIX commands:

/	Specifies the base, root directory of the entire file system.
.	Specifies the current working directory.
..	Specifies the parent directory of current working directory.
&	Execute the specified command in another process.

Important UNIX commands and keystroke conventions

Remember to do things in the following sections that are in bold. Do things in the right order, without skipping anything. That way it will work! Some may seem repetitive, but remember, repetition fosters learning.

Getting help in any OS can be very important. UNIX provides a text-based help system called man pages. You use man pages by typing the command "man" followed by the name of the command that you want help on. Most commands have online documentation available through the man pages. Give the command "**man**

tcsh" to see how the man command pages you through the manual pages of the help system, and to read about the T shell:

```
> man tcsh
```

Press the space bar to page through man pages; type the letter "**q**" for quit to return to your command prompt. A helpful option to man is "`-k`," which searches through man page titles for specified words:

> man -k batch Gets you the title lines for every command with the word batch in the title.

Another help system, "`info`," may be installed as well. Use it similarly to man, i.e. "`info cmd`."

When an account is created, your home directory environment variable, "`$HOME`," is created and associated with that account. In any tree structured file system the concept of where you are in the tree is very important. There are two ways of specifying where things are. You can refer to things relative to your current directory or by its complete 'path' name. When the complete path name is given by beginning the specification with a slash, the current position in the directory tree is ignored. To find the complete path in the SCS file system to your current directory (`$HOME` at the point) type the command "**pwd**" (print working directory)

```
> pwd
/a/fs.csit.fsu.edu/u7/users/thompson
```

This UNIX command shows you where you are presently located on the server. It displays the complete UNIX path specification (this always starts with a slash) for the directory structure of your account. Also notice that UNIX uses forward slashes (`/`) to differentiate between subdirectories, not backward slashes (`\`) like MS-DOS. The pwd command can be used at any point to keep track of your location. Several commands for working with your directory structure follow:

> **pwd**	Print working directory. Shows where you are at in the file system. Very useful when you get confused. (Also see "`whoami`" if you're really confused!)
> **ls**	Shows (lists) your files' names, i.e. the contents of the current directory
> **ls -l**	Shows files' names in extended (long) format with size, ownership, and permissions.
> **ls -al**	Shows all files including dot systems files in your directory in the long format.
> **mkdir newdir**	Makes a new directory named "`newdir`" in your current directory.
> **cd newdir**	Move down into a directory named "**newdir**" from your current directory.
> **cd**	Move back into your home directory from anywhere (with most shells).
> **rmdir newdir**	Removes a subdirectory from your current directory. Directory must be empty.

To list the files in your home directory, use the "**ls**" command. There are many options to the ls command. Check them out by typing "**man ls**". The most useful options are the "**-l**," "**-a**," and "**-t**" options. The command "**ls -l**" will list the files and directories in your current directory in the 'long' form with extended information. The "**-a**" option displays 'all' files, even files with a period as the first character in their name, a

UNIX convention to hide important system files from normal listing. The "**-t**" option displays files ordered by 'time,' with the most recent first. These options can be used in any combination, e.g. "**ls –alt**."

This convention has lead to a number of special configuration files with periods as the first character in their name. Some of these files are executed automatically when a user logs in, much like "AUTOEXEC.BAT" and "CONFIG.SYS" are executed in MS-DOS upon log in. On many UNIX systems there is a file executed upon every login called "**.**login" and another one that sets up the shell environment called "**.**cshrc" or "**.**tcshrc". Don't mess with these files until you are quite comfortable with UNIX. Three examples of the ls command in my SCS account follow:

```
> ls
1EFT.pdb   Desktop   Latitude  packages     RepFiles
Cn3D_User  dumpster  mail      public_html  SPDBV

> ls -l
total 291
-rw-r--r--    1 thompson faculty    279370 Feb 22  2005 1EFT.pdb
drwxr-xr-x    2 thompson faculty        96 Sep 14  2004 Cn3D_User
drwx------    3 thompson faculty        96 Apr 21 13:52 Desktop
drwxr-xr-x    3 thompson faculty        96 Mar 29  2001 dumpster
-rw-r--r--    1 thompson faculty      7401 Jan 17  2000 Latitude
drwx------    2 thompson faculty        96 Aug 28  2001 mail
drwxr-xr-x    4 thompson faculty        96 Apr 26 21:49 packages
drwxr-xr-x    2 thompson faculty      1024 Aug  1 21:38 public_html
drwxr-xr-x    2 thompson faculty      9216 Mar 17  2004 RepFiles
drwxr-xr-x    7 thompson faculty        96 Jan 17  2000 SPDBV

> ls -a
.                   .fonts.cache-1        .Latitude          .pine-debug2
..                  .fullcircle           .launchrc          .pine-debug3
1EFT.pdb            .gconf                .login             .pine-debug4
.acrobat            .gconfd               mail               .pinerc
.acrorc             .gimp                 .mailcap           .profile
.addressbook        .gnome                .mc                public_html
.addressbook.lu     .gnome2               .mcop              .qt
.adobe              .gnome2_private       .MCOP-random-seed  .recently-used
.AmiraRegistry      .gnome-desktop        .mcoprc            RepFiles
.bash_history       .gnp                  .metacity          .rhn-applet.conf
.bashrc             .gtkrc                .mh_profile        SPDBV
Cn3D_User           .gtkrc-1.2-gnome2     .mime.types        .ssh
.cshrc              .gtkrc-kde            .mozilla           .ssh2
.DCOPserver_cdburn  .gxedit               .mplayer           .sversionrc
Desktop             .gxedit.applications  .nautilus          .user60.rdb
dumpster            .gxedit.last          .netscape          .viminfo
.e-conf             .history              .netscape6         .wmrc
.ee                 .ICEauthority         .newsrc-news       .xauth
.emacs.d            .java                 .openoffice        .Xauthority
.enlightenment      .kde                  packages           .xftcache
.esd_auth           .kderc                .pauphistory       .xsession.bak
.first_start_kde    Latitude              .pine-debug1       .xsession-errors
```

In the output from "**ls -l**" additional information regarding the file permissions, owner of the file, size, modification date, and file name is shown. In the output from "**ls -a**" all those 'dot' systems files are now seen. Nearly all OSs have some way to customize your login environment with editable configuration files;

UNIX uses these dot files. An experienced user can put commands in dot files to customize their individual login environment.

Another example of the ls command, along with output redirection is shown below. Issue the following command to generate a file named "**SCS.program.list**" that lists program names in long format located in the SCS directory "**/usr/common/i686-linux/bin/**," many of which are phylogenetic in nature:

```
> ls -l /usr/common/i686-linux/bin/ > SCS.program.list
```

Rather than scrolling the ls output to the screen, the output is redirected into the file "SCS.program.list"

An environment variable, your $PATH, tells your account what directories to look in for programs; /usr/common/i686-linux/bin/ above, is in your path, so you can run any of the programs in "SCS.program.list" by just typing its name. You can see your complete path designation by using the command "**echo**," along with **$PATH**, which 'echoes' its meaning to the screen. Each path, of the several listed, is separated by a colon:

```
> echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/condor/bin:/usr/common/i686-linux/bin:/u
sr/X11R6/bin
```

Subdirectories are generally used to group files associated with one particular project or files of a particular type. For example, you might store all of your PAUP* command block templates in a directory called "PAUPblocks." The "**mkdir**" command is used to create directories and the "**cd**" command is used to move into directories. The special placeholder file "**..**" allows you to move back up the directory tree. Note its use below with the "cd **..**" command to go back up one level to the parent of the current directory:

```
> mkdir PAUPblocks

> ls
1EFT.pdb   Desktop   Latitude  packages   public_html  SPDBV
Cn3D_User  dumpster  mail      PAUPblocks  RepFiles

> cd PAUPblocks

> pwd
/a/fs.csit.fsu.edu/u7/users/thompson/PAUPblocks

> cd ..

> pwd
/a/fs.csit.fsu.edu/u7/users/thompson/
```

After the "cd **..**" command pwd shows that we are 'back' in the home directory. Note that with most shells "cd" all by itself from anywhere in your account will take you all the way home. Next let's look at several basic commands that affect the file system and access files, rather than directories:

> **cat SCS.program.list**	Displays contents of the file "**SCS.program.list**" to screen without pauses; also concatenates files (appends one to another), e.g: "`cat file1 file2 > file3`" or "`cat file1 >> file2`."
> **more SCS.program.list**	Shows the contents of the file "**SCS.program.list**" on the terminal one page at a time; press the space bar to continue. Type a "`?`" when the scrolling stops for viewing options. Type "`/pattern`" to search for "`pattern`." (less is often available; it's more powerful than more — silly computer systems humor).
> **head SCS.program.list**	Shows the first few lines of the file "**SCS.program.list**," optionally –N lines from the top of the file.
> **tail SCS.program.list**	Show the last few lines of the file "**SCS.program.list**," optionally –N lines from the bottom of the file.
> **wc SCS.program.list**	Counts the number of characters, words, and lines in specified file.
> **cp SCS.program.list tmp1**	Copies the file "**SCS.program.list**" to the file "**tmp1**." Any previous contents of a file named "**tmp1**" are lost.
> **mv SCS.program.list tmp2**	Renames (moves) the file "**SCS.program.list**" to the file "**tmp2**." Any previous contents of a file "**tmp2**" are lost, and "**SCS.program.list"** no longer exists.
> **cp tmp2 PAUPblock**	Since "**PAUPblocks**" is a directory name not a file name, this command copies the specified file, "**tmp2**," into the specified directory, "**PAUPblocks**," keeping the file name intact. Use the "`-R`" recursive option to copy all files down through a directory structure.
> **rm tmp2**	Deletes (removes) the file "**tmp2**" in the current directory.
> **rm PAUPblocks/tmp2**	Deletes (removes) the file "**tmp2**" in the directory "**PAUPblocks**." It is unrecoverable and permanently gone!

More commands that deal with files (but don't do these):

`rm -r somedir`	Removes all the files, and subdirectories of a directory and then removes the directory itself — very convenient, very useful, and very dangerous. Be careful!
`chmod somefile`	Changes the permissions of a file named "`somefile`." See "`man chmod`" and also "`man chown`" for further details.
`lpr somefile`	Prints the specified file on the default printer. You need to specify a particular print queue with the "`-P`" option to send it elsewhere.

An example using the SCS /usr/common/i686-linux/bin/ program list again is shown here. This time the ls output is piped to the more command rather than redirected into a file:

> **ls -l /usr/common/i686-linux/bin/ | more**

A useful command that allows searching through the contents of files for a pattern is called grep. The first parameter to grep is a search pattern; the second is the file or files that you want searched. For example, if you have a bunch of different Nexus data files whose file names all end with the word "`.nex`" in several different subdirectories, all one level down, and you wanted to find the one that has the word zebra in it, you could "`grep zebra */*.nex`." Use the following variation of the grep command to see all the programs in our SCS program list that have the word "pro" in them:

> **`grep pro tmp1`** Show the lines in the file "**`tmp1`**" that contain the specified pattern, here the word "**`pro`**" (these are all PHYLIP protein sequence specific phylogenetics programs).

Another file searching command, "**`find`**," looks not within files' contents, but rather at their names, to help you find files that are lost in your directory structure. Its syntax is a bit strange, not following the usual rules:

> **`find . -name '*tmp*'`** Finds files from the current directory (.) down containing the word tmp anywhere within its filename. Note that the single quotes (') are necessary for wild card expansion to occur with the find command.

Commands for looking at the system, other users, your sessions and jobs, and command execution follow:

> **`uptime`** Shows the time since the system was last rebooted. Also shows the "load average". Load average indicates the number of jobs in the system ready to run. The higher the load average the slower the system will run.

> **`w`** (or `who`) Shows who is logged in to the system doing what.

> **`top`** Shows the most active processes on the entire machine and the portion of CPU cycles assigned to running processes. Press "**`q`**" to quit.

> **`ps`** Shows your current processes and their status, i.e. running, sleeping, idle, terminated, etc. Use "**`man ps`**" as options vary widely, see especially the -a, -e, -l, and -f options).

> **`ps –U user`** Perhaps (user is you) the most useful ps option — show me all of MY processes!

Some more process commands that we won't be using today are shown here:

`at` Submit script to the at queue for execution later.

`bg` Resumes a suspended job in background mode.

`fg` Brings a background job back into interactive mode.

And the command to change your password (which won't be needed today either):

`passwd` Change your login password (but on SCS systems ssh to sun2, as per https://www.csit.fsu.edu/twiki/bin/view/TechHelp/LoginShell).

Usually it is best to leave programs using a quit or exit command; however, occasionally it's necessary to terminate a running program. Here are some useful commands for bailing out of programs:

< Ctrl c >	Abort (cancel) a running process (program); there's no option for restarting it later.
< Ctrl d >	Terminate a UNIX shell, i.e. exit present control level and close the file. Use "logout" or "exit" to exit from your top-level login shell.
< Ctrl z >	Pause (suspend) a running process and return the user to the system prompt. The suspended program can be restarted by typing "fg" (foreground). If you type "bg" (background), the job will also be started again, but in background mode.
> kill –9 psid	Kills a process with the given process ID using the "sure kill" option. The PSID number is obtained using some variation of the ps command.

Text editing — the good, the bad, and the ugly

Text editing is often a necessary part of computing. This is never that much fun, but it can be very, very important. You can use your own favorite word processor like MS Word, if you insist, but be sure to "Save As" "Text Only" with "Line Breaks," and specify UNIX line breaks, if you have the choice. Native word processing format contains a whole bunch of binary control data specifying format and fonts and so forth; the UNIX OS can't read it at all. Saving as text only avoids this problem. Using an ASCII text editor like BBEdit on a Mac avoids the binary problem, but you still need to be careful to save with UNIX style line breaks.

Editing files on your own personal computer and then using them on a different computer is a two-step process though. After all the editing is done, the file will need to be transferred with scp or sftp to the UNIX server where it will be used. Therefore, it makes sense to get comfortable with at least one UNIX text editor. That will avoid the file transfer step, saving some hassle and time. There are several around, including many driven though a GUI, but minimally I recommend learning pico. It's description, along with two alternatives follow. Launch pico of the tmp1 file with the following command:

> **pico tmp1**	A simple text editor provided with the pine mailer. It is quite appropriate for general text editing, but is not present on all UNIX systems (it can be installed on any UNIX system). This is a very easy to use editor with a command banner presenting a menu of Ctrl Key command options. Type some sample text into the file, then press < **Ctrl x** > to exit, reply "y" for yes to save the file, and then accept or modify the file's name.

Two other command line UNIX editors are described below, but do not use these today:

`vi file`	The default UNIX text editor. This comes with all versions of UNIX and is extremely powerful, but it is quite difficult to master. I recommend avoiding it entirely unless you are interested in becoming a true UNIX expert.
`emacs file`	This is a very nice alternative text editor available on many UNIX machines. This editor is also quite powerful but not nearly as difficult to learn as vi.

File transfer — getting stuff from there to here, and here to there

You will often need to move files back and forth between different computers. Remember scp from the Introduction. That's the primary secure way to move files around within the Internet. I never use removable media like floppy or Iomega disks, or CDs, or USB drives anymore. I just copy files between machines over the Internet. The commands in the following table provide simple access to a small subset of UNIX networking capabilities (host refers to a computer's fully qualified Internet name or number):

`ftp host`	File transfer protocol. Allows a limited set of commands (dir, cd, put, get, help, etc.) for moving files between machines. Note: insecure method, so often restricted to particular servers that allow "anonymous ftp" only. See sftp and scp as an alternative.
`scp`	Secure copy file, syntax: "`scp file user@host:path`" or "`scp user@host:path file`." Good for moving one or a few files at a time.
`sftp`	Secure file transfer protocol. Allows same subset of commands as ftp, but through an encrypted connection. Good for moving lots of files.
`telnet host`	Provides an insecure terminal connection to another Internet connected host (discouraged and often disabled!). See ssh for a secure alternative.
`ssh user@host`	Connect to a host computer using a secure, encrypted protocol. This is often the only allowed way to interactively log onto a remote computer.

Let's practice with scp to give you a feel for its syntax. The example that we will use here is actually quite silly and not at all required, since all SCS general use UNIX computers share the same user disk space (through NFS mounting). That means that all of your files exist in your SCS account independent of which machine you log onto (except for some special use servers). Regardless, issue the following command to see how command line scp works (your SCS user ID replaces "user" in the examples below). Note the required colon, "**:**" in the syntax. In its most simple form:

```
> scp tmp1 user@pamd.csit.fsu.edu:
```

You'll most likely get the same sort of authenticity question as when you first connected to phoenix; answer "**yes**" and then supply your SCS password. This will put a copy of the file "**tmp1**" from the current, 'local' machine, phoenix, to your home directory on the scp connected, 'remote' machine, pamd. Let's do it the other way 'round now, that is, from a remote pamd to a local phoenix, with a few extra twists:

```
> scp user@pamd.csit.fsu.edu:tmp1 PAUPblocks/tmp3
```

OK, what does this command do? It logs you onto pamd and looks for a file named tmp1 in your home directory there. Then it copies that file into your PAUPblocks directory on the local phoenix server that you're already logged onto and changes its name to tmp3. scp also supports a "`-r`" recursive option so that it can be used to secure copy down through the contents of a directory structure. Simple enough. Got it?

Microsoft Windows machines and Macs often have a GUI form of scp/sftp installed. In the MS Windows world this may be called secure file transfer client, and on OS X Macs a great little free program named Fugu can be used (http://rsug.itd.umich.edu/software/fugu/). Let's get rid of those tmp files now before proceeding. Note that you can remove more than one file specification at a time. Issue the following command:

```
> rm tmp* */tmp*
```

The UNIX 'background'

The whole point of today's workshop session was to get comfortable enough at the UNIX command line to run large phylogenetic analyses in the UNIX server environment. Well, with large analyses that will run a few to many days, there's no way that you'll want to keep an active terminal session with a running process in the 'foreground' that whole time. That particular terminal session wouldn't be able to do anything else. Furthermore, and much more importantly, any interruption in Internet access, or any crash of your local computer would cause the job to abort on the server! Depending on the program being run, you may not get any, to only partial results. Therefore, big server-based jobs should always be run in the UNIX 'background.' There are many ways to submit this type of job. I'll discuss a general method that should work with most any program. A generalized complete command line is shown below; not all parts are absolutely necessary:

```
> nohup nice cmd –options parameters < input > output >& logfile &
```

Let's look at each part in turn. First notice that the command line actually has three commands in a row, nohup, nice, and cmd, i.e. whatever program you want to run in the background. The "`nohup`" command guarantees that subsequent commands will not be interrupted by any type of 'hangup' signal and will continue to run after you log out. The second part, "`nice`" is a 'nice' way to run background processes — it allows your program to use 100% of the server CPU when nobody else is using it, but less if it would interfere with the performance of another user's session. It does this by reducing the process's scheduling priority (by adding a default 10 to the existing priority out of range with a high of –20 and low of 19). The third command, "`cmd`," is whatever CPU intensive program and its options and parameters that you want to run in the background, e.g. in subsequent workshop sessions you'll learn how to run PAUP* and MrBayes this way. The actual command lines for PAUP* and MrBayes background runs, as well how you use the CONDOR distributed batch queuing system for spreading your job over more than one node of a cluster will be taught.

The remainder of the command line specifies input, output, and error direction, and submits the entire thing to the UNIX background queue. As seen earlier, for programs that don't accept input and output specifications without redirection, "<" directs input, for instance, a command script of some sort, into the preceding command, and ">" directs program output into some file. You haven't yet seen ">`&`." This strange combination, unique to the T and C shell, tells the program to put any normal screen trace and any standard error report into a file, here named "`logfile`." The final ampersand, "`&`," submits the whole command to the background. You can happily log off the server and go about your merry way — the job will run without you.

Check on the status of jobs with top or variations of ps. The output will be waiting for you when it's finished.

At the end of any UNIX terminal session issue the command "**`exit`**" (or "`logout`") to log you off the remote server! Usually servers will disconnect if you force the connection to close, but sometimes this can cause hung session problems. Be safe, save often, develop good computer habits, and always log out.

Conclusion

Today's tutorial should have provided you with the basics necessary to get about in the UNIX OS. You should now feel somewhat comfortable at the UNIX command line, at least enough so as to maintain your file and directory structure in your new SCS account, and to move files between that account and other computers that you may have access to. Account maintenance is your own responsibility. Keep your account clean and organized. Name files logically with names that you will recognize later on. Use consistent naming conventions, particularly filename extensions (that part after the dot). If you can't find your files or figure out what is what sometime later down the road, you only have yourself to blame.

UNIX is not the easiest OS to learn. Have patience, ask questions, and don't get down on yourself just because it doesn't seem as easy as other OSs. The power and flexibility of UNIX is worth the extra effort. Plus, UNIX is the *de facto* standard OS for most scientific computing, so the effort will not be wasted.

Ultimately computers only do what they have been programmed to do. Their accuracy entirely depends on the software being used, the data being analyzed, and the manner in which it is used. In biocomputing this means that the accuracy and relevance of your results depends on your understanding of the strengths, weaknesses, and intricacies of both the software employed, and of the biological system being studied.

Acknowledgements

I offer special thanks to Charles Severance for providing much of the material that my UNIX tutorial was based upon (http://www.msu.edu/~systems/ucgdocuments/misc/unixtut.html). I'd also like to acknowledge Susan Jean Johns, my former colleague and supervisor at the Center for Visualization, Analysis and Design at Washington State University, now at the University of California San Francisco, for providing some of the introductory material for the tutorial — thank you Susan for teaching me so very much over my early years in bioinformatics.