BioInformatics: A SeqLab Introduction

Bioinformatics is tough — use a comprehensive, server-based technology to cope with the data!

July 16 & 17, 2008, a GCG¥ Wisconsin Package™ SeqLab® tutorial supplement for Fort Valley State University.

Author and Instructor: Steven M. Thompson

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

BioInformatics: A SeqLab introduction

It's a pretty new field, been around not quite thirty years or so, called various, often misunderstood names, that are largely subsets of one another — computational molecular biology, biocomputing, bioinformatics, sequence analysis, molecular modeling, and most lately genomics and proteomics. But what does it all mean? One way to think about it is the reverse biochemistry analogy — biochemists no longer have to begin a research project by isolating and purifying massive amounts of a protein from its native organism in order to characterize a particular gene product. Rather, now scientists can amplify a section of some genome based on its similarity to other genomes, sequence that piece of DNA, and, using sequence analysis tools, infer all sorts of functional, evolutionary, and, perhaps even, structural insight into a gene within it, and then, most likely, go on to clone that gene, express the gene product, and finally purify the protein. The process has come full circle. The computer has become an important tool to be used at the beginning and throughout a research project in assisting experimental design, not just a number cruncher used at the end of the process. This is only possible because of modern computational speed and power and the tremendous growth of the molecular databases. Biocomputing's explosive growth is reflected in and largely a result of the increase in the level of computational processing power available, along with a concurrent exponential growth of the molecular sequence databases. GenBank doubles in size every18 months! GenBank version 165, April 2008, has 9,172,350,468 bases, from 85,500,730 reported sequences, and this doesn't include the 110,500,961,400 bases in 26,931,049 sequences within the Whole Genome Shotgun (WGS) database.

First, a prelude — my definitions

Much confusion abounds in the area, even concerning the names of the disciplines themselves. The terms are often bantered about with little regard to what they really mean. Here's my slant on the situation. All are interdisciplinary by nature, combining elements of computer and information science, mathematics and statistics, and chemistry and biology. Each has elements of one another. Biocomputing and computational biology are the most encompassing terms and can be considered synonyms. They both describe using computers and computational techniques to analyze a biological system, whether that is a biomolecular primary sequence or tertiary structure, or a metabolic pathway, or even a complex system such as the interactions of populations within an ecological niche.

Bioinformatics necessarily intersects with this concept in that it describes using computational techniques to access, analyze, and interpret the biological information in databases. However, these databases can be the traditionally considered nucleic and amino acid sequence databases as well as three-dimensional molecular structure databases, but can even include such disparate data collections as medical records or population statistics. Therefore, bioinformatics is a type of biocomputing but also includes topics such as medical informatics that is not usually considered a part of computational biology.

Within bioinformatics the subdiscipline of sequence analysis has a clearly defined scope. It is the study of biological molecular sequence data for the purpose of inferring the function, interactions, evolution, and

perhaps structure of biological molecules. Molecular modeling can also be considered a type of bioinformatics, though it often isn't. It is necessarily a subdiscipline of computational structural biology, but uses the methodology and techniques of that discipline as well sequence analysis' similarity searching and alignment algorithms. That is why it is often referred to as "homology modeling."

Genomics is the subdiscipline of bioinformatics that is concerned not with individual molecular sequences, but rather with sequences on a genomic scale. That is, genomics analyzes the context of genes or complete genomes (the total DNA content of an organism) and transcriptomes (the total mRNA content of an organism) within and across genomes. Proteomics can be considered the subdivision of genomics concerned with analyzing the complete protein complement, i.e. the proteome, of organisms, both within and between different organisms.

Structural genomics is the acquisition and analysis of the complete set of three-dimensional structure coordinate data for an organism's entire proteome (or a representative set thereof). Through these types of analyses it may eventually be possible to predict a completely unknown protein's structure and function just based on its deduced molecular sequence. Obviously this could be an incredible boost to the drug-design process and could go a long way toward curing many disease processes. We have come a long way in structural prediction, but are still a long way from this goal. The comparative method is crucial to all these methods but, perhaps most obvious and key to genomics and proteomics.

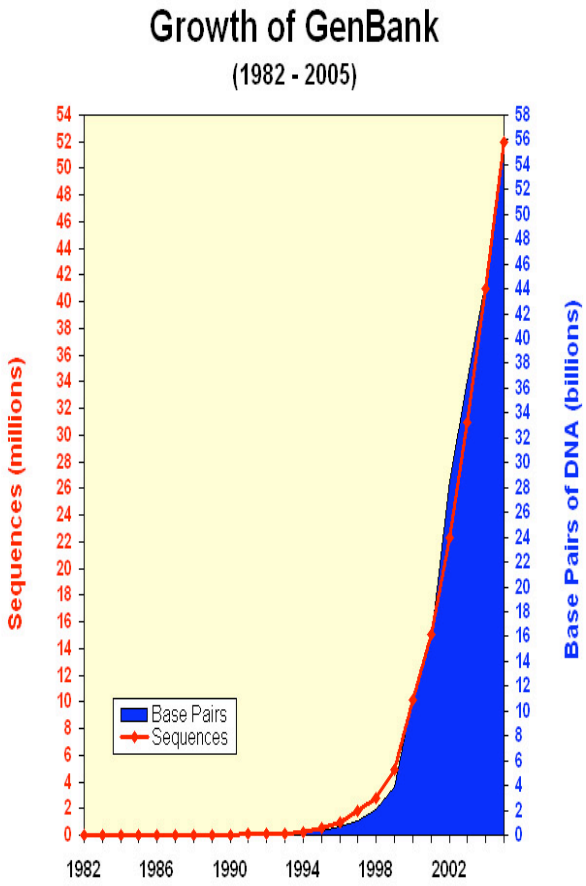
I. Databases: Content and Organization

The first genome sequenced was *Haemophilus influenzae*, at the Johns Hopkins University School of Medicine (Fleischmann, et al, 1995). The International Human Genome Sequencing Consortium announced the completion of a "Working Draft" of the human genome in June 2000 (Lander, et al., 2001); independently that same month, the private company Celera Genomics announced that it had completed the first assembly of the human genome (Venter, et al., 2001). As of April 2008, over 50 Archaea, over 600 Bacteria, around 20 Eukaryote complete genomes, and about 200 Eukaryote assemblies were represented, depending on your definition of complete (not even NCBI agrees with itself on this point!), and not counting the more than 3,000 virus and viroid genomes available. Among the Eurkaryota are a cryptomonad, *Guillardia theta*, flagellate, *Leishmania major*, apicomplexan, *Plasmodium falciparum* and *yoelli*, red alga, *Cyanidioschyzon merolae*, microsporidium, *Encephalitozoon cuniculi*, baker's yeast, *Saccharomyces cerevisiae*, fission yeast, *Schizosaccharomyces pombe*, nematode, *Caenorhabditis elegans*, mosquito, *Anopheles gambiae*, honeybee, *Apis mellifera*, fruit fly, *Drosophila melanogaster*, sea squirt, *Ciona intestinalis*, zebrafish, *Danio rerio*, chimpanzee, *Pan troglogdytes*, human, *Homo sapiens*, mouse, *Mus musculus*, rat, *Rattus norvegicus*, thale cress, *Arabidopsis thaliana*, oat, *Avena sativa*, soybean, *Glycine max*, barley, *Hordeum vulgare*, tomato, *Lycopersicon esculentum*, rice, *Oryza sativa,* wheat, *Triticum aestivum*, and corn, *Zea mays*. (somewhat conflicting genome statistics at NCBI on several of their own Web pages: http://www.ncbi.nlm.nih.gov/genomes/VIRUSES/viruses.html,

http://www.ncbi.nlm.nih.gov/genomes/leuks.cgi, http://www.ncbi.nlm.nih.gov/genomes/lproks.cgi, and http://www.ncbi.nlm.nih.gov/genomes/static/gpstat.html).

Over half of the genes in many of these organisms have predicted functions based solely on previously studied bacterial genes, the comparative method in practice. Numerous worldwide genome projects have kept the data coming at alarming rates. The primary nucleotide database in the U.S.A., NCBI's GenBank, has staggering growth statistics (http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html):

Year	BasePairs	Sequences
1982	680,338	606
1983	2,274,029	2,427
1984	3,368,765	4,175
1985	5,204,420	5,700
1986	9,615,371	9,978
1987	15,514,776	14,584
1988	23,800,000	20,579
1989	34,762,585	28,791
1990	49,179,285	39,533
1991	71,947,426	55,627
1992	101,008,486	78,608
1993	157,152,442	143,492
1994	217,102,462	215,273
1995	384,939,485	555,694
1996	651,972,984	1,021,211
1997	1,160,300,687	1,765,847
1998	2,008,761,784	2,837,897
1999	3,841,163,011	4,864,570
2000	11,101,066,288	10,106,023
2001	15,849,921,438	14,976,310
2002	28,507,990,166	22,318,883
2003	36,553,368,485	30,968,418
2004	44,575,745,176	40,604,319
2005	56,037,734,462	52,016,762
2006	69019290705	64893747
2007	83874179730	80388382



A. What are primary sequences?

Remember biology's Central Dogma: DNA → RNA → protein. Primary refers to one dimensional — all of the "symbol" information written in sequential order necessary to specify a particular biological molecular entity, be it polypeptide or polynucleotide. The symbols are the one letter alphabetic codes for all of the biological nitrogenous bases and amino acid residues and their ambiguity codes (see the nice explanatory table at http://virology.wisc.edu/acp/CommonRes/SingleLetterCode.html). Biological carbohydrates, lipids, and structural information are not included within this sequence; however, much of this type of information is available in the annotation associated with primary sequences in the databases.

B. What are sequence databases?

These databases are an organized way to store the tremendous amount of sequence information that is piling up at exponential rates, as seen above. Each database has its own specific format, and access to this information is most easily handled through various software packages and interfaces, either on the World Wide Web (WWW) or otherwise.

In Bethesda, Maryland, United States, the National Center for Biotechnology Information (NCBI, http://www.ncbi.nlm.nih.gov/), a division of the National Library of Medicine (NLM), at the National Institutes of Health (NIH), supports and distributes the GenBank primary nucleic acid sequence database and the GenPept CDS (CoDing Sequence) translations database. They also maintain the derivative RefSeq genome, transcriptome, and proteome sequence databases, the preliminary data Whole Genome Shotgun project depository (WGS) database, and they provide access to data in other sequence databases maintained by the rest of the worldwide supporters. The other primary database organization in the United States is the National Biomedical Research Foundation (NBRF, http://pir.georgetown.edu/nbrf/), an affiliate of Georgetown University Medical Center. They maintain the Protein Identification Resource (PIR, http://pir.georgetown.edu/) database of polypeptide sequences, which has now been consolidated into the UniProt database (the Universal Protein Resource, http://www.uniprot.org/).

The European Bioinformatics Institute, EBI (http://www.ebi.ac.uk/) in Hinxton, Cambridge, United Kingdom, a part of the European Molecular Biology Laboratory (EMBL http://www.embl-heidelberg.de/) in Heidelberg, Germany, maintains the EMBL nucleic acid sequence database. The Swiss Institute of Bioinformatics, SIB, at ExPASy (the Expert Protein Analysis System, http://www.expasy.org/) in Geneva, Switzerland, and EBI jointly support the excellently annotated Swiss-Prot protein sequence database, as well as the minimally annotated TrEMBL (Translations from EMBL — those EMBL translations not yet in Swiss-Prot) protein sequence databases. UniProt, a coalition of EBI, SIB, and PIR, contains sequences from all of the protein databases.

Additional, less well known, sequence databases include sites with the military, with private industry, and in Japan (the DNA Data Bank of Japan, DDBJ, http://www.ddbj.nig.ac.jp/). In most cases data is openly exchanged between the databases so that many sites 'mirror' one another. This is particularly true with GenBank, EMBL, and DDBJ; there is never a need to look in all three places. The same is now true with the creation of UniProt — the best one stop shop for protein sequence data and annotation.

C. What information do they contain, how is it organized, and how is it accessed?

Sequence databases are often mixtures of ASCII and binary data; however, they usually aren't true relational or object oriented data structures. Many expensive proprietary ones are though, and some public domain ones are MySQL. It's a complicated mess with little standardization. Typical sequence databases contain several very long ASCII text files that contain information of all the same type, such as all of the sequences themselves, versus all of the title lines, or all of the reference sections. Binary files usually help 'tie together'

all of the files by providing indexing functions. Software specific routines, as exemplified by genome browsers and text search tools, are by far the most convenient method to successfully interact with these databases.

Nucleic acid databases (and TrEMBL) are split into subdivisions based on taxonomy (historical). Protein databases are often split into subdivisions based on the level of annotation that the sequences have.

Annotation sections include extremely valuable information — reference author and journal citations, organism and organ of origin, and the features table. The features table lists all sorts of important regulatory, transcriptional and translational (CDS coding sequence), catalytic, and structural sites, depending on the database. Actual sequence data usually follows the annotation in most formats.

Becoming familiar with the general format of sequence files for the type of software you want to use can save a lot of grief. Unfortunately most databases and many different software packages have conflicting format requirements. Fortunately there are many excellent format converters available such as ReadSeq (Gilbert, 1993 and 1999). However, most sequence analysis software requires that you specify a proper sequence name and/or database identifier. These are usually discovered with some sort of text searching program, either on the WWW, e.g. Entrez (Schuler, et. al, 1996) or SRS (Sequence Retrieval System, Etzold and Argos, 1993), or with some type of a dedicated local program. This brings up a point, locus names versus accession numbers. The LOCUS, ID, and ENTRY names category in the various databases are different than the Accession number category. Each sequence is given a unique accession number upon submission to the database. This number allows tracking of the data when entries are merged or split; it will always be associated with its particular data. Entry names may change; accession numbers are forever; they just pile up, primary becomes secondary, *ad infinitum*.

D. What changes have occurred in the databases — history and development?

The first well recognized sequence database was Margaret Dayhoff's *Atlas of Protein Sequence and Structure* begun in the mid-sixties (Dayhoff, et al., 1965–1978), which later became PIR (George, et al., 1986). GenBank began in 1982 (Bilofsky, et al., 1986), EMBL in 1980 (Hamm and Cameron, 1986). They have all been attempts at establishing an organized, reliable, comprehensive, and openly available library of genetic sequences. Databases have long-since outgrown a hardbound atlas. They have become huge and have evolved through many changes. Changes in format over the years are a major source of grief for software designers and program users. Each program needs to be able to recognize particular aspects of the sequence files; whenever they change, it's liable to throw a wrench in the works. People have argued for particular standards such as XML (called BSML, Bioinformatics Sequence Markup Language, for sequence data), but it's almost impossible to enforce. NCBI's ASN.1 format (Abstract Syntax Notation One, an International Standards Organization [ISO] data representation format with inter-platform operability) and its Entrez interface were another attempt to circumvent these frustrations. Entrez, EMBL's SRS, found on the WWW at all EMBL outstations, and the Wisconsin Package's LookUp derivative of SRS all search for text in, interact with, and allow users to browse the sequence databases. Both SRS and Entrez provide 'links' to

associated databases so that you can jump from, for instance, a chromosomal map location, to a DNA sequence, to its translated protein sequence, to a corresponding structure, and then to a MedLine reference, and so on. They are very helpful!

E. What other types of bioinformatics databases are used?

Specialized versions of sequence databases include sequence pattern databases such as restriction enzyme (e.g. http://rebase.neb.com/) and protease (e.g. http://merops.sanger.ac.uk/) cleavage sites, promoter sequences and their binding regions (e.g. http://www.gene-regulation.com/pub/databases.html and http://www.epd.isb-sib.ch/), and protein motifs (e.g. http://us.expasy.org/prosite/) and profiles (e.g. http://www.sanger.ac.uk/Software/Pfam/); and organism or system specific databases such as the sequence portions of ACeDb (A *C. elegans* Database http://www.acedb.org/), FlyBase (*Drosophila* database http://flybase.bio.indiana.edu/), SGD (*Saccharomyces* Genome Database http://www.yeastgenome.org/), GDB (the Human Genome Database, http://gdbwww.gdb.org/), and RDP (the Ribosomal Database Project http://rdp.cme.msu.edu/). Many of these organism specific databases present their data in the context of a genome map browser (e.g. the University of California, Santa Cruz, bioinformatics group's human genome browser, http://genome.ucsc.edu/, and the Ensembl project, http://www.ensembl.org/, jointly hosted by the Welcome Trust Sanger Institute and the European Bioinformatics Institute). Map browsers attempt to tie together as many data types as possible using a physical map of a particular genome as a framework that a user can zoom in or out on in order to see more or less detail of any particular loci.

Two other types of databases are commonly accessed in bioinformatics: reference and three-dimensional structure. Reference databases run the gamut from OMIM (Online Mendelian Inheritance In Man, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=OMIM), that catalogs human genes and phenotypes, particularly those associated with human disease states, and their excellent descriptive database Gene (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene), that provides a 'genecentric' view of completely sequenced genomes, to PubMed access of MedLine bibliographic references (the National Library of Medicine's citation and author abstract bibliographic database of over 4,800 biomedical research and review journals, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed). Other databases that could be put in this class include things like proprietary medical records databases and population studies databases.

Finally, the Research Collaboratory for Structural Bioinformatics (RCSB), a consortium of institutions: Rutgers University, the State University of New Jersey; the San Diego Supercomputer Center, University of California, San Diego; and the University of Wisconsin-Madison; supports the three-dimensional structure Protein Data Bank (PDB http://www.rcsb.org/pdb/). The National Institute of Health maintains "Molecules To Go" at http://molbio.info.nih.gov/cgi-bin/pdb as a very easy to use interface to PDB. And NCBI maintains the MMDB (Molecular Modeling DataBase) (http://www.ncbi.nlm.nih.gov/Structure/MMDB/mmdb.shtml) that contains all of the experimentally determined structures from PDB. Other three-dimensional structure databases include the Nucleic Acid Databank at Rutgers (NDB http://ndbserver.rutgers.edu/) and the proprietary Cambridge small molecule Crystallographic Structural Database (CSD http://www.ccdc.cam.ac.uk/products/csd/).

II. So how does one do bioinformatics?

A. Bioinformatics and the Internet: the World Wide Web

Often bioinformatics is done on the Internet through the WWW. This is possible and easy and fun, but, besides being a bit too easy too get sidetracked upon . . . the Web can not readily handle large datasets or large multiple sequence alignments. These types of datasets quickly become intractable. You'll know you're there when you try. In spite of that . . .

Some of my favorite WWW sites for molecular biology and bioinformatics follow below:

Site	URL (Uniform Resource Locator)	Content
National Center Biotech' Info'	http://www.ncbi.nlm.nih.gov/	databases/analysis/software
PIR/NBRF	http://pir.georgetown.edu/	protein sequence database
ProteinDataBank	http://www.rcsb.org/pdb/	3D mol' structure database
Molecules To Go	http://molbio.info.nih.gov/cgi-bin/pdb/	3D protein/nuc' visualization
IUBIO Biology Archive	http://iubio.bio.indiana.edu/	database/software archive
Univ. of Montreal Genomics	http://megasun.bch.umontreal.ca/	database/software archive
Japan's GenomeNet Server	http://www.genome.ad.jp/	databases/analysis/software
European Mol' Bio' Lab'	http://www.embl-heidelberg.de/	databases/analysis/software
European Bioinformatics Inst'	http://www.ebi.ac.uk/	databases/analysis/software
The Sanger Institute	http://www.sanger.ac.uk/	databases/analysis/software
Swiss Institute Bioinformatics	http://www.expasy.ch/	databases/analysis/software
Human Genome DataBase	http://www.gdb.org/	Human Genome Project
Stanford Genomic Resource	http://genome-www.stanford.edu/	various genome projects
Inst. for Genomic Research	http://www.tigr.org/	microbial genome projects
HIV Sequence Database	http://hiv-web.lanl.gov/	HIV epidemeology seq' DB
The Baylor Search Launcher	http://searchlauncher.bcm.tmc.edu/	sequence search launcher
Pedro's BioMol Res' Tools	http://www.public.iastate.edu/~pedro/research_tools.html	extensive bookmark list
Harvard Bio' Laboratories	http://golgi.harvard.edu/BioLinks.html	nice bookmark list
BioToolKit	http://www.biosupplynet.com/cfdocs/btk/btk.cfm	annotated molbio tool links
Felsenstein's PHYLIP site	http://evolution.genetics.washington.edu/phylip.html	phylogenetic inference
The Tree of Life	http://tolweb.org/tree/	overview of all phylogeny
Ribosomal Database Project	http://rdp.cme.msu.edu/index.jsp	databases/analysis/software
PUMA2 Metabolism	http://compbio.mcs.anl.gov/puma2/cgi-bin/index.cgi	metabolic reconstructions
BIOSCI/BIONET	http://net.bio.net/	biologists' news groups
Access Excellence	http://www.accessexcellence.org/	biology teaching and learning
CELLS alive!	http://www.cellsalive.com/	animated microphotography

B. So what are the alternatives . . . ?

Desktop software solutions — public domain programs are available, but . . . complicated to install, configure, and maintain. User must be pretty computer savvy. So, commercial software packages are available, e.g. MacVector, Sequencher, DNAsis, DNAStar, etc.

But . . . license hassles, big expense per machine, and database access all complicate matters!

C. Therefore, server-based solutions (e.g. the GCG Wisconsin Package) — UNIX server computers

These offer very fast, convenient database access on local server disks, and connections can be made from any networked terminal or workstation anywhere! Public domain solutions also exist, but now a very cooperative systems manager needs to maintain everything for users. Commercial, server-based software have one license fee for an entire institution, rather than individual licenses for every computer.

But . . . operating system and command line hassles.

1. Communications software

Most all computer systems will have some type of a WWW browser available, be it Microsoft's Explorer, Netscape's Navigator, Mozilla's Firefox, KDE's Konqueror, Opera Software ASA's Opera, Apple's Safari, on *ad infinitum*, it doesn't matter. You can use whatever is on the machine. Unfortunately a Web browser alone is not enough for serious biocomputing. More often than not you will need to directly connect to a server computer using a command line, "terminal," window where you can directly interact with the server computer's operating system. The 'old way' to do this was with a common program called telnet. However, telnet is an unsecure program from which smart hackers can 'sniff' connection 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 operating systems but doesn't come with pre-OS X Macintosh or any Micrsoft Windows machines and, therefore, must be installed on those platforms separately in order to do most server-based biocomputing.

2. X graphics

Furthermore, since ssh is strictly a non-graphical terminal program, and since all Web browsers' graphics capability is inadequate for the truly interactive graphics that much biocomputing software requires, another type of graphical system needs to be present on the computer that you use for much biocomputing. That graphical interface is called the X Window System (*a.k.a.* X11). It was developed at MIT (the Massachusetts Institute of Technology) in the 1980's, back in the early days of UNIX, as a distributed, hardware independent way of exchanging graphical information between different UNIX computers. Unfortunately the X worldview is a bit backwards from the standard client/server computing model. In the standard model a local client, for instance a Web browser, displays information from a file on a remote server, for instance a particular WWW site, also called a Uniform Resource Locator (URL). In the world of X, an X-server program on the machine that you are sitting at (the local machine) displays the graphics from an X-client program that could be located on either your own machine or on a remote server machine that you are connected to. Confused yet?

X-server graphics windows take a bit of getting used to in other ways too. For one thing, they are only active when your mouse cursor is in the window. And, rather than holding mouse buttons down, to

activate X items, just <click> on the icon. Furthermore, X buttons are turned on when they are pushed in and shaded, sometimes it's just kind of hard to tell. Cutting and pasting is real easy, once you get used to it — select your desired text with the left mouse button, paste with the middle. Finally, always close X Windows when you are through with them to conserve system memory, but don't force them to close with the X-server software's close icon in the upper right- or left-hand window corner, rather, always, if available, use the client program's own "File" menu "Exit" choice, or a "Close," "Cancel," or "OK" button.

Nearly all UNIX computers, including Linux, but not including Macintosh OS X, include a genuine X Window System in their default configuration. Microsoft Windows computers are often loaded with X-server emulation software, such as the commercial programs XWin32 or eXceed, to provide X-server functionality. Macintosh computers prior to OS X required a commercial X solution; often the program MacX or eXodus was used. However, since OS X Macintoshes are true UNIX machines, they can use one of a variety of free open source packages such as XDarwin to provide true X Windowing. Perhaps the best X solution for Max OS X is Apple's own X11 package distributed for free from their support pages: http://www.apple.com/downloads/macosx/apple/x11formacosx.html.

3. Text editing

At some point you will have to edit a file; text editing is often a necessary part of computing. This is never that much fun, but always very important. The UNIX operating system always has vi installed. It's a part of the operating system and is very powerful, but quite intimidating. Emacs or pico (or nano) are often provided as alternatives. Or you can use your own favorite desktop word processing software like Microsoft Word, if you would like, followed by file transfer. Just be sure to "Save As" "Text Only" with "Line Breaks," but don't be surprised if you have subsequent line break problems, unless you can specify UNIX style line breaks.

Native word processing format contains binary control data in it specifying format and so forth; the UNIX operating system can't read it. Saving as text avoids this problem. Editing this way is a two-step process though. After the editing is done, the file needs to be transferred to the UNIX server. Therefore, it makes sense to get comfortable with at least one UNIX text editor. That will avoid the file transfer step, saving some hassle. There are several around, including many driven though a graphical user interface (GUI), but minimally I recommend learning pico (or its nearly identical clone nano).

4. File transfer

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' unsecure 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 is called sftp and scp, for

'secure file transfer protocol' and 'secure copy' respectively. It's also included in all modern UNIX operating systems, but not in pre-OS X Macintoshes, nor in Microsoft Windows, so it has to be installed on those computers separately.

III. A basic guide to UNIX for neophytes

Because this is all somewhat confusing to newcomers, here's a UNIX tutorial that we won't take the time to go through today, but I encourage you to do so at some point. I stole this tutorial from the Internet and have extensively modified it over the years for bioinformatics use. I am indebted to the countless, but unnamed, contributors — I apologize for my lack of credit giving and flagrant copyright infringement. Hundreds of users worldwide are grateful; thank you.

The original UNIX operating system (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. One of the most popular variations is RedHat Linux. 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.

All UNIX OSs 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 and using a "shell" terminal window. However, becoming familiar with some basic UNIX commands will make your computing experience much less frustrating. The shell program is your command line 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. Among numerous UNIX command line guides available on the Internet, there's a very good beginning UNIX tutorial at http://www.ee.surrey.ac.uk/Teaching/Unix/, if you would like to see an alternative approach to what I 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, Microsoft 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.

A. Generalities

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.

B. 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 file system to your current directory type the command "**`pwd`**" (print working directory). My server's example follows:

```
> pwd
/home/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**," "**-t,**" and "**-a**" options. These options can be used in any combination, e.g. "**ls –alt**." The "**-l**" option will list the files and directories in your current directory in a 'long' form with extended information. The "**-t**" option displays files ordered by 'time,' with the most recent first. 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.

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

```
> ls
bin         EF1a-primitive.DNA.nex  mail      prime.csh   SPDBV
Cn3D_User   EF1a-Tu.DNA.nex         molevol   ribo_files  tutorials
cut.csh     gcg                     nsmail    seqlab      working
db_info     Latitude                patterns  snap_files

> ls -l
total 228
drwxr-xr-x    4 thompson gcg          4096 Oct 25 12:07 bin
drwxr-xr-x    2 thompson gcg          4096 Oct 25 17:42 Cn3D_User
-rwxr-xr-x    1 thompson gcg           162 Feb 13  2003 cut.csh
drwxr-xr-x    2 thompson gcg          4096 Jan 16  2001 db_info
-rw-r--r--    1 thompson gcg         82710 Apr 28  2005 EF1a-
primitive.DNA.nex
-rw-r--r--    1 thompson gcg         63796 Apr 28  2005 EF1a-Tu.DNA.nex
drwxr-xr-x    2 thompson gcg          4096 Oct 11 17:32 gcg
-rw-r--r--    1 thompson gcg          7401 Jan 17  2000 Latitude
drwx------    2 thompson gcg          4096 Jan 27  2005 mail
drwxr-xr-x    9 thompson gcg          4096 Aug 12  2004 molevol
drwx------    2 thompson gcg          4096 Oct 18 12:33 nsmail
drwxr-xr-x    4 thompson gcg          4096 Jun  3  1999 patterns
-rw-r--r--    1 thompson gcg           538 Apr  1  2004 prime.csh
drwxr-xr-x   15 thompson gcg          4096 Oct 16  2001 ribo_files
drwxrwxr-x    2 thompson gcg          4096 Oct 25 11:53 seqlab
drwxr-xr-x    4 thompson gcg          4096 Oct 25 12:05 snap_files
drwxr-xr-x    7 thompson gcg          4096 Jan 17  2000 SPDBV
drwxr-xr-x    7 thompson gcg          4096 Oct 19 14:05 tutorials
drwxr-xr-x   17 thompson gcg          4096 Nov  8 14:11 working

> ls -a
.                       .forward   .mailcap     .pauphistory     snap_files
..                      gcg        .mc          .pinerc          SPDBV
bin                     .history   .mime.types  prime.csh        .ssh
Cn3D_User               .java      molevol      ribo_files       tutorials
cut.csh                 Latitude   .mozilla     seqlab           working
db_info                 .Latitude  .ncftp       .seqlab-history  .wp
EF1a-primitive.DNA.nex  .login     nsmail       .seqlab-mendel   .Xauthority
EF1a-Tu.DNA.nex         mail       patterns     .seqmerge
```

In the output from "**ls -l**" additional information regarding file permissions, owner, size, and modification date 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 "**program.list**" that lists all of the file names in long format located in your server's "**/usr/local/bin/**" directory:

```
> ls -l /usr/local/bin/ > program.list
```

Rather than scrolling the ls output to the screen, this command redirects it into the file "program.list"

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

```
> echo $PATH

      /usr/local/gcg/License_Pack/Linux_2_Intel_32/exe:/usr/local/gcg/License_Pa
      ck/bin:/usr/local/gcg/bin:/usr/local/bin:/bin:/usr/bin:/usr/java/j2sdk1.4.
      0_01/bin:/usr/local/Cn3D-4.1:/usr/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 memorandums in a directory called "memo." 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. Check out its use below with the cd command to go back up to the parent of the current directory:

```
> mkdir memo

> ls
bin      gcg        mail  molevol   ribo_files  snap_files  temp.ps   working
db_info  login.bak  memo  patterns  seqlab      temp.epsf   tutorials

> cd memo

> pwd
/home/thompson/memo

> cd ..

> pwd
/home/thompson
```

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

> **cat program.list**	Displays contents of the file "**program.list**" to screen without pauses; also concatenates files (appends one to another), e.g: "`cat file1 file2 > file3`" or "`cat file1 >> file2`."
> **more program.list**	Shows the contents of the file "**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 program.list**	Shows the first few lines of the file "**program.list**," optionally –N lines from the top of the file.
> **tail program.list**	Show the last few lines of the file "**program.list**," optionally –N lines from the bottom of the file.
> **wc program.list**	Counts the number of characters, words, and lines in specified file, "**program.list**."
> **cp program.list tmp1**	Copies the file "**program.list**" to the file "**tmp1**." Any previous contents of a file named "**tmp1**" are lost.
> **mv program.list tmp2**	Renames (moves) the file "**program.list**" to the file "**tmp2**." Any previous contents of a file "**tmp2**" are lost, and "**program.list"** no longer exists.
> **cp tmp2 memo**	Since "**memo**" is a directory name not a file name, this command copies the specified file, "**tmp2**," into the specified directory, "**memo**," 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 memo/tmp2**	Deletes (removes) the file "**tmp2**" in the directory "**memo**." 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 (and extensive) details.
`lpr somefile`	Prints the specified file on a default printer. Specify a particular print queue with the "`-P`" option to send it elsewhere.

Another example using the /usr/local/bin/ program list is shown here. This time the ls output is piped to the more command rather than redirected into a file:

> **ls -l /usr/local/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 data files whose file names all end with the word "`.data`" in several different subdirectories, all one level down, and you wanted to find the one that has the word zebra within it, you could "`grep zebra */*.data`." Use the following variation of the grep command to see all the programs in our Mendel 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.

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 >` Do not use this to stop a job! It pauses (suspends) a running process and returns 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.

C. 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. As mentioned earlier, you can use your own favorite word processor like Microsoft 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 Macintosh 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 (or nano). 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:

`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.
`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.

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

You will often need to move files back and forth between different computers. Remember scp from section II.4. 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: unsecure 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.

I'll illustrate scp to give you a feel for its syntax. Note the required colon "**:**" in the command. In its simplest form:

```
> scp file user@somemachine.somewhere:
```

You'll most likely get the same sort of authenticity question as when you first connect to a machine with ssh; answer "yes" and then supply your password. This will put a copy of the file from your current, 'local' machine to your home directory on the scp connected, 'remote' machine.

Let's do it the other way 'round now, that is, from a remote server to a local machine, with a few extra twists:

```
> scp user@somemachine.somewhere:somedir/file somedir/somefile
```

OK, what does this command do? It logs you onto a remote machine and looks for a file named “file” in your “somedir” directory there. Then it copies that file into your “somedir” directory on your local machine that you’re already logged onto and it changes its name from “file” to “somefile.” 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 Macintoshes often have a GUI form of scp/sftp installed. In the Microsoft Windows world this may be called secure file transfer client, and on OS X Macintoshes 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*
```

E. Using X between different UNIX computers

These are the bare-minimum instructions necessary for connecting to a UNIX host computer from another UNIX computer using X. Not all commands are necessary in all cases, as they are often set by your account environment; however, I’ll supply a complete set. In most cases fully qualified Internet names can be used in these procedures, however, depending on local name servers, you may need to specify IP numbers. A fictitious example host machine, zen.art.motorcycle.com, has the following name and number:

```
zen.art.motorcycle.com        999.999.99.99
```

You will need to know your own machine’s name and/or number as well as the host's.

Log on to your UNIX workstation account in the customary manner. Depending on the workstation, you may want to specify an xterm terminal window if your terminal window is not already an xterm. On most systems:

Optional: `> /usr/bin/X11/xterm &`

On Solaris: `> /usr/openwin/bin/xterm &`

Following UNIX X commands with an ampersand, "`&`," is helpful so that they are run in the background in the new window in order to maintain control of the initial window. Some helpful options supported in most versions of xterm are “`-ls`” so that your login script is read, “`-sb -sl 100`” to give you a 100 line scroll back capability, “`-tn vt220`” to take advantage of vt220 terminal features, and “`-fg Bisque -bg MidnightBlue`” to give you nice light colored characters on a dark blue background.

Then at your workstation’s UNIX prompt, authorize X access to the host with the xhost command:

```
> xhost +zen.art.motorcycle.com                    (should not be necessary)
```

Next connect to the host with the ssh command; e.g:

```
> ssh -X thompson@zen.art.motorcycle.com
```
(-capital X sets the X environment for you)

This should produce a login window. Log in as usual, then, if necessary, issue the following command on the host to setup the X environment (for the c shell and its derivatives), where your_IP_node_name represents the Internet name or number of the workstation that you are sitting at:

```
Host> setenv DISPLAY your_IP_node_name:0
```
(again, should not be necessary)

It is best to run commands from an X terminal window rather than from a default console window as is sometimes created by a remote connection. Therefore, after setting up your environment, an option is to launch xterm by minimally issuing the xterm command to the host (as discussed above, many options are available).

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.

IV. The Genetics Computer Group — the Accelrys Wisconsin Package for Sequence Analysis

GCG began in 1982 on a VAX/VMS computer in Oliver Smithies' Genetics Department laboratory at the University of Wisconsin, Madison; and then starting in 1990 it became a private company; which was acquired by the Oxford Molecular Group, United Kingdom, in 1997; and then by Pharmacopeia Inc., United States, in 2000; and then in 2004 Accelrys, San Diego, California, left Pharmacopeia to become an independent entity. The suite is now exclusively a UNIX package and contains around a 150 programs designed to work in a "toolbox" fashion. That is, several simple programs used in succession can lead to sophisticated results. Most importantly, the package has 'internal compatibility,' i.e. once you learn to use one program, all programs can be run similarly, and, the output from many programs can be used as input for other programs. It is used all over the world, for more than 20 years, by more than 30,000 scientists at over 950 institutions worldwide, so learning it here will most likely be useful at any of several places you may end up at.

A. Specifying sequences GCG style and logical terms!

To answer the perplexing GCG question, "What sequence(s)?" In order of increasing power and complexity, four methods:

1. The sequence is in a local GCG single sequence format file (SSF) in your UNIX account. This sequence file can be anywhere in your account as long as you supply an appropriate 'path' so that the program can find the file. The sequence file can have any name but it is best to use extensions that tell you what type of molecule it is, e.g. `.seq` and `.pep` (`my.pep` or `~user/subdir/my.seg`). Use the program '`reformat`' to convert 'raw' text format files to GCG format.

```
This is a small example of 'raw' GCG single sequence format.
```

```
Always put some documentation on top, so in the future you
can figure out what it is you're dealing with!  Two periods
always separate that documentation from the actual data.

..

ACTGACGTCACATACTGGGACTGAGATTTACCGAGTTATACAAGTATACAGATTTAATAGCATGCGATCCCATG
GGA
```

Next the clean GCG format single sequence file after the '`reformat`' command:

```
!!NA_SEQUENCE 1.0

This is a small example of GCG single sequence format.
Always put some documentation on top, so in the future
you can figure out what it is you're dealing with!  The
line with the two periods is converted to the checksum line.

example.seq  Length: 77  July 21, 1999 09:30  Type: N  Check: 4099  ..

1  ACTGACGTCA CATACTGGGA CTGAGATTTA CCGAGTTATA CAAGTATACA
51  GATTTAATAG CATGCGATCC CATGGGA
```

2. The sequence is in a local GCG database in which case you 'point' to it by using any of the GCG database logical names. These names make sense, and are either the name of the database or an abbreviation thereof. Subcategory logical names can be used for nucleotide databases, such as rodent. A colon, "**:**" always sets the logical name apart from either an accession code or a proper identifier name or a wildcard expression, and they are case insensitive. Some examples follow: GenBank:EctufBT, gb:x57091, UniProt:EFTu_Ecoli, and uni:p02990 all refer to elongation factor Tu sequences in *Escherichia coli*, the first two nucleotide, the last two protein. If the database uses consistent naming conventions, then you can use a wild card to specify all of a particular type of sequence. This works particularly well in SwissProt; e.g. SW:EFTu_* specifies all of the EFTu sequences in SwissProt. Because sequences are in local GCG databases, it is seldom necessary to put individual sequences in your account, that only takes up disk space.

Logical terms for the GCG Wisconsin Package (terms may vary at different installations)

Sequence databases, nucleic acids:

GENBANKPLUS:*	all of GenBank plus EST, HTC, and GSS
GBP:*	all of GenBank plus EST, HTC, and GSS
GENBANK:*	all of GenBank except EST, HTC, and GSS
GB:*	all of GenBank except EST, HTC, and GSS
TAGS:*	GenBank EST, HTC, and GSS subdivisions
EST:*	GenBank EST (expressed sequence tags) subdivision

Sequence databases, amino acids:

GENPEPT:*	GenBank CDS translations
GP:*	GenBank CDS translations
UNIPROT:*	PIR, SwissProt, and TrEMBL
UNI:*	PIR, SwissProt, and TrEMBL
SWISSPROTPLUS:*	PIR, SwissProt, and TrEMB
SWP:*	PIR, SwissProt, and TrEMBL

GSS:*	GenBank GSS (genome survey sequences) subdivision	UNIPROTSPROT:*	SwissProt (fully annotated)
HTC:*	GenBank HTC (high throughput cDNA) subdivision	UNISPROT:*	SwissProt (fully annotated)
HTG:*	GenBank HTG (high throughput genomic) subdivision	SWISSPROT:*	SwissProt (fully annotated)
PATENT:*	GenBank patent subdivision	SW:*	SwissProt (fully annotated)
PAT:*	GenBank patent subdivision	UNIPROTTREMBL:*	preliminary EMBL translations
STS:*	GenBank (sequence tagged sites) subdivision	UNITREMBL:*	preliminary EMBL translations
SYNTHETIC:*	GenBank synthetic subdivision	SPTREMBL:*	preliminary EMBL translations
SY:*	GenBank synthetic subdivision	SPT:*	preliminary EMBL translations
UNANNOTATED:*	GenBank unannotated subdivision		
UN:*	GenBank unannotated subdivision		
BACTERIAL:*	GenBank bacterial subdivision		
BA:*	GenBank bacterial subdivision		
INVERTEBRATE:*	GenBank invertebrate subdivision		
IN:*	GenBank invertebrate subdivision		
OTHERMAMMAL:*	GenBank other mammalian subdivision		
OM:*	GenBank other mammalian subdivision		
OTHERVERTEBRATE:*	GenBank other vertebrate subdivision		
OV:*	GenBank other vertebrate subdivision		
PHAGE:*	GenBank phage subdivision		
PH:*	GenBank phage subdivision		
PLANT:*	GenBank plant subdivision		
PL:*	GenBank plant subdivision		
PRIMATE:*	GenBank primate subdivision		
PR:*	GenBank primate subdivision		
RODENT:*	GenBank rodent subdivision		
RO:*	GenBank rodent subdivision		
VIRAL:*	GenBank viral subdivision		
VI:*	GenBank viral subdivision		

3. The sequence is in a GCG format multiple sequence file, either an MSF (multiple sequence format) file or an RSF (rich sequence format) file. The difference is that MSF files contain only the sequence names and sequence symbol characters, whereas RSF files contain names, annotation, and actual sequence data. As in GCG single sequence format, it is always best to retain the suggested GCG extensions, msf or rsf, in order for you to easily recognize what type of file they are without having to look, though it is not required and they could just as well be named Joe.Blow. To specify sequences contained in a GCG multiple sequence file, supply the file name followed by a pair of braces, "{}," containing the sequence specification. For example, to specify all of the sequences in an alignment of elongation 1α and Tu factors, one may use a naming system such as the following: ef1a-tu.msf{*}. Furthermore, one can point to individual members of the alignment or subgroups by specifying their name within the braces, e.g. EF1a-Tu.rsf{eftu_ecoli} to point just to the *E coli* sequence or EF1a-Tu.rsf{eftu_*} to point at all of the EfTu's as long as you use a sequence naming convention that retains this convention.

4. The most powerful method of specifying sequences is in a GCG "list" file. This file can have any name though it is convenient to use the extension ".list" to help identify list files in your account. It is merely a list of other sequence specifications, and can even contain other list files within it. The convention to use

a GCG list file in a program is to precede it with an at sign, "@." Furthermore, you can supply attribute information within list files to specify something special about the sequence. This is especially helpful with length attributes that can restrict an analysis to specific portions of a sequence, and can be seen in the example below:

```
!!SEQUENCE_LIST 1.0

An example GCG list file of many elongation 1α and Tu factors follows.  As
with all GCG data files, two periods separate documentation from data.  ..

my-special.pep          begin:24    end:134
SwissProt:EfTu_Ecoli
Ef1a-Tu.msf{*}
/usr/accounts/test/another.rsf{ef1a_*}
@another.list
```

B. SeqLab — a brief history — Steve Smith's GDE + GCG's WPI

While working on bacterial ribosomal RNA phylogenies with Walter Gilbert and Carl Woese, Steve Smith realized the need for a comprehensive multiple sequence editor. Nothing existed at the time that satisfied him, so he invented one. In addition to providing the vital editing function, it also served as a menuing system to external functions such as PHYLIP routines and Clustal alignments. He called it the "Genetic Data Environment" (Smith, et al., 1994). Many people were very impressed, and he made it freely available, though it was designed just for Sun workstations. Coincidentally GCG realized the need for some sort of a 'point-and-click' environment for their system. They were losing lots of business, only being able to provide a command line interface. Therefore, they started trying to develop a GUI for the Wisconsin Package. They called it the "Wisconsin Package Interface." Nobody was impressed — it was a terrible attempt. It only provided a menu to their programs, hardly anything more than the "–check" option they've always had. So they did a natural and very smart thing. They hired Steve Smith away from Millipore, where he had newly moved, into their company, so that he could merge his GDE with their WPI. The offspring was SeqLab, and, thank goodness, they threw away the acronyms. As 'they' say "The rest is history" and once more GCG's customers are (generally) happy.

SeqLab, an X-based GUI to the Wisconsin Package — and some illustrative examples: Glutathione Reductase, G-protein coupled TM7 receptors, primate prions, Human Papilloma Virus L1 major coat protein, Major Histocompatibility Class II, Vicilin seed storage proteins, and Elongation Factor 1α/Tu.

C. And what's the deal with the new "+" programs?

Quoting directly from the GCG Program Manual:

"Advantages of Plus "+" Programs:

√ Plus programs are enhanced to be able to read sequences in a variety of native formats such as GCG RSF, GCG SSF, GCG MSF, GenBank, EMBL, FastA, SwissProt, PIR, and BSML without conversion.

√ Plus programs remove sequence length restriction of 350,000 bp.

If you do not need these features and wish to have more interactivity, you might wish to seek out and run the original program version."

D. For more information do the accompanying tutorial

Computers only do what they have been programmed to do. Your answers entirely depend on the software being used, the data being analyzed, and the manner in which it is used. In scientific biocomputing research, this means that the accuracy and relevancy of your results depends on your understanding of the strengths, weaknesses, and intricacies of both the software and data employed, and, probably most importantly, of the biological system being being studied.

References

Bairoch, A. (1991) The Swiss-Prot Protein Sequence Data Bank. *Nucleic Acids Research* **19**, 2247–2249.

Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer Jr., E.F., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T., and Tasumi, M., (1977) The Protein Data Bank: A computer-based archival file for macromodel structures. *Journal of Molecular Biology* **112**, 535–542.

Bilofsky, H.S., Burks, C., Fickett, J.W., Goad, W.B., Lewitter, F.I., Rindone, W.P., Swindell, C.D., and Tung, C.S. (1986) The GenBank™ Genetic Sequence Data Bank. *Nucleic Acids Research* **14**, 1–4.

Dayhoff, M.O., Eck, R.V., Chang, M.A. and Sochard, M.R. (1965) *Atlas of Protein Sequence and Structure*, Vol. 1. National Biomedical Research Foundation, Silver Spring, MD, U.S.A.

Etzold, T. and Argos, P. (1993) SRS — an indexing and retrieval tool for flat file data libraries. *Computer Applications in the Biosciences* **9**, 49–57.

Fleischmann, R.D., Adams, M.D., White, O., Clayton, R.A., Kirkness, E.F., Kerlavage, A.R., Bult, C.J., Tomb, J.F., Dougherty, B.A., Merrick, J.M., et al., (1995) Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* **269**, 496–512.

George, D.G., Barker, W.C., and Hunt, L.T. (1986) The Protein Identification Resource (PIR). *Nucleic Acids Research* **14**, 11–16.

Genetics Computer Group (GCG®), (Copyright 1982-2007) *Program Manual for the Wisconsin Package®*, version 11.1, http://www.accelrys.com/products/gcg/ Accelrys Inc., San Diego, California, U.S.A.