Last modified August 19, 2015

Elementary Sequence Analysis

Brian Golding, Dick Morton and Wilfried Haerty

Department of Biology
McMaster University
Hamilton, Ontario
L8S 4K1

These notes are in Adobe Acrobat format (they are available upon request in other formats) and they can be obtained from the website http://helix.biology.mcmaster.ca/courses.html. Some of the programs that you will be using in this course and which will be run locally can be found at http://evol.mcmaster.ca/p3S03.html.

The "blue text" should designate links within this document while the "red text" designate links outside of this document. Clicking on the latter should activate your web browser and load the appropriate page into your browser. If these do not work please check your Acrobat reader setup. The web links are accurate to the best of our knowledge but the web changes quickly and we cannot guarantee that they are still accurate. The links designated next to the JAVA logo,  , require that JAVA be installed on your computer.

These notes are used in Biology 3S03. The purpose of this course is to introduce students to the basics of bioinformatics and to give them the opportunity to learn to manipulate and analyze DNA/protein sequences. Of necessity only some of the more simple algorithms will be examined.

The course will hopefully cover . . .

- databases of relevance to molecular biology.
- some common network servers/sites that provide access to these databases.
- methods to obtain sequence analysis software and data.
- methods of sequence alignment.
- methods of calculating genetic distance.
- methods of phylogenetic reconstruction.
- methods for detecting patterns and codon usage.
- methods for detecting gene coding regions.

The formal part of the course will consist of two approximately one hour lectures each week. Weekly assignments will be be provided to practice and explore the lecture material. In addition there will be an optional tutorial to help students with these assignments or other problems. These assignments will be 40% of your grade and three, in class quizzes will make up the remainder.

We would appreciate any comments, corrections or updates regarding these notes.

Golding@McMaster.CA Morton@McMaster.CA HaertyW@McMaster.CA

This document was entirely constructed with pdfLaTeX. Enormous kudos to all those that make this great software free to everyone.

Table of Contents in Brief

In order to speed download, I place here links to the individual chapters in pdf format. The contents of these are shown on the following 'Contents' pages but note that the links will function only for the individual chapter included here.

Preliminaries

Basic Unix

Genomics

Databases

Sequence File Formats

Sequence Alignment

Distance Measures

Database Searching

Reconstructing Phylogenies

Pattern analysis

Exon analysis

Contents

1 Preliminaries **1**

1.1 Resources . . . 1

1.1.1 Electronic Resources . . . 1

1.1.2 Textbooks . . . 2

1.1.3 Journal sources . . . 7

1.2 Biological preliminaries . . . 10

1.2.1 Some notes on terminology . . . 10

1.2.2 Letter Codes for Sequences . . . 11

2 Computer skills preliminaries **13**

2.1 UNIX Operating Systems . . . 13

2.1.1 Logging on/off . . . 14

2.1.2 UNIX File System . . . 14

2.1.3 Commands . . . 17

2.1.4 Help . . . 19

2.1.5 Redirection . . . 20

2.1.6 Shells . . . 20

2.1.7 Special 'hidden' files . . . 21

2.1.8 Background Processes . . . 21

2.1.9 Utilities . . . 22

2.1.10 Editors . . . 22

2.2 Exchange among computers . . . 24

2.2.1 ssh . . . 24

2.2.2 Mail . . . 24

2.3 Scripts-Languages . . . 25

2.4 Obtaining LINUX . . . 25

3 Genomics **27**

3.1 Where the data comes from . . . 27

3.2 How DNA is sequenced . . . 27

3.3 First Generation Methods . . . 28
3.4 The reality of sequencing includes errors . . . 32
3.5 From sequence to genome . . . 33
3.6 Second (Next) Generation Sequencing . . . 37
3.7 Paired sequences . . . 43
3.8 Third Generation Sequencing . . . 44
3.9 Upcoming Sequencing Technologies . . . 45
3.10 Types of sequencing . . . 46
3.10.1 Exome sequencing . . . 46
3.10.2 RAD-tag seq . . . 47
3.10.3 BAsE-seq . . . 47
3.10.4 RNA-seq . . . 47
3.10.5 BS-seq . . . 48
3.10.5.1 TAB-seq . . . 48
3.10.5.2 NOMe-seq . . . 49
3.10.6 Regulatory sequencing: DNase-seq/FAIRE-seq . . . 49
3.10.7 ChIP-seq . . . 49
3.10.7.1 CLIP-seq . . . 49
3.10.8 Hi-C . . . 50
3.11 Other kinds of biological data . . . 50
3.11.1 Microarrays . . . 51
3.11.2 Mass spectrometry methods . . . 56
3.11.3 Textual information . . . 56

4 Databases 59
4.1 Introduction . . . 59
4.2 N.C.B.I. . . . 62
4.3 E.M.B.L. . . . 67
4.4 D.D.B.J. . . . 68
4.5 SwissProt . . . 69
4.6 Organization of the entries . . . 71
4.7 Other Major Databases . . . 73
4.8 Remote Database Entry retrieval . . . 76
4.8.1 Entrez . . . 76
4.8.2 NCBI retrieve . . . 77
4.8.3 EMBL get . . . 79
4.8.4 Others . . . 80
4.9 Reliability . . . 80

5 Sequence File Formats **83**
5.1 Genbank/EMBL . . . 83
5.2 FASTA . . . 85
5.3 FASTQ . . . 86
5.4 SAM/BAM format . . . 87
5.5 Stockholm format . . . 88
5.6 GDE . . . 90
5.7 NEXUS . . . 92
5.8 PHYLIP . . . 93
5.9 ASN . . . 94
5.10 BSML format . . . 97
5.11 PDB file format . . . 97

6 Sequence Alignment **103**
6.1 Dot Plots . . . 103
6.1.1 The Exact Way . . . 103
6.1.2 Identity Blocks . . . 105
6.2 Alignments . . . 113
6.2.1 The Needleman and Wunsch Algorithm . . . 113
6.2.2 The Smith-Waterman Algorithm . . . 116
6.3 Testing Significance . . . 117
6.4 Gaps and Indels . . . 120
6.4.1 "Natural" Gap Weights - Thorne, Kishino & Felsenstein . . . 120
6.5 Multiple Sequence Alignments . . . 121

7 Distance Measures **125**
7.1 Nucleotide Distance Measures . . . 125
7.1.1 Simple counts as a distance measure . . . 125
7.1.2 Jukes - Cantor Correction . . . 126
7.1.3 Kimura 2-parameter Correction . . . 128
7.1.4 Tamura - Nei Correction . . . 128
7.1.5 Uneven spatial distribution of substitutions . . . 129
7.1.6 Synonymous - nonsynonymous substitutions . . . 130
7.2 Amino acid distance measures . . . 130
7.2.1 PAM Matrices . . . 131
7.2.2 BLOSUM Matrices . . . 133
7.2.3 GONNET Matrix . . . 134
7.3 Gap Weighting . . . 135

8 Database Searching **137**
8.1 Are there homologues in the database? . . . 137
8.1.1 FASTA . . . 137
8.1.1.1 Instructions . . . 137
8.1.1.2 FASTA output . . . 139
8.1.1.3 FASTA format . . . 142
8.1.1.4 Statistical Significance . . . 144
8.1.2 BLAST . . . 145
8.1.2.1 BLAST output . . . 146
8.1.2.2 BLAST format . . . 150
8.1.3 MPsrch . . . 152
8.1.3.1 MPsrch output . . . 153
8.1.3.2 MPsrch format . . . 155
8.2 BLOCKS . . . 156
8.2.1 BLOCKS output . . . 157
8.2.2 Getting the Block . . . 158
8.3 SSearch . . . 164
8.4 Why you should routinely check your sequence . . . 164

9 Reconstructing Phylogenies **165**
9.1 Introduction . . . 165
9.1.1 Purpose . . . 165
9.1.2 Trees of what . . . 165
9.1.3 Terminology . . . 167
9.1.4 Controversy . . . 169
9.2 Distance Methods . . . 169
9.3 Parsimony Methods . . . 171
9.4 Other Methods . . . 174
9.4.1 Compatibility methods . . . 174
9.4.2 Maximum Likelihood methods . . . 174
9.4.3 Method of Invariants . . . 175
9.4.4 Quartet Methods . . . 176
9.5 Consensus Trees . . . 178
9.6 Bootstrap trees . . . 178
9.7 Warnings . . . 181
9.8 Available Packages . . . 182
9.9 PHYLIP . . . 186
9.9.1 PHYLIP Contents . . . 186

10 Pattern Analysis **199**
10.1 Base Composition: first order patchiness . . . 199
10.1.1 Genome Patchiness . . . 199
10.2 Dinucleotide Composition: second order patchiness . . . 200
10.3 Strand Asymmetry . . . 201
10.3.1 Chargaff's Rules . . . 201
10.3.2 Replication Asymmetry . . . 202
10.3.3 Transcriptional Asymmetry . . . 203
10.3.4 Codon Selection . . . 204
10.4 Simple Sequence Repeats . . . 204
10.5 Sequence Complexity . . . 204
10.5.1 Information Theory . . . 204
10.5.2 Sequence Window Complexity . . . 206
10.6 Finding Pattern in DNA Sequences . . . 207
10.6.1 Consensus Sequences . . . 207
10.6.2 Matrix Analysis of Sequence Motifs . . . 208
10.6.3 Sequence Conservation and Sequence Logos . . . 209

11 Exon Analysis **213**
11.1 Open Reading Frames . . . 213
11.2 Gene Recognition . . . 213
11.2.1 Splice Sites . . . 214
11.2.2 Codon Usage . . . 215
11.2.3 Gene Prediction Software . . . 218
11.2.4 Hidden Markov Models (HMM) . . . 219
11.2.5 Comparison of Programs . . . 219

Chapter 2

Computer skills preliminaries

Bioinformatics is about the manipulation of biological data and how to turn that data stream into biological knowledge. Due to the large amounts of data, this cannot be done by hand but fortunately the explosion of biological data streams in the last decades have been matched by a revolution in computer technology. In this section I give the briefest of introductions to the UNIX operating system. Part of the philosophy of the UNIX system is to provide the user with basic tools that accomplish one abstracted and atomized task well rather than providing a polished piece of software that accomplishes a single end result. In research, it is useful to have the tools that can be combined in order to construct a new result rather than to use a software suite that accomplishes only what it's original designer envisioned. As a result, UNIX is an operating system that is said to "wear well", meaning that the more you learn the more you can do, the more you can accomplish and the more there is to learn.

My purpose in this chapter is to quickly introduce a novice to UNIX computers to such a level that they can perform useful work. For this course 'work' is being defined as an ability to enter, to search out, but mostly to manipulate and to analyze sequences and produce information that is biologically relevant.

2.1 UNIX Operating Systems

In the past I have covered three different types of computer operating systems but for some years now this diversity has been limited to just one; UNIX systems. Why UNIX? There are several reasons but perhaps the most important from a novice's viewpoint is simply that UNIX is what is being used. Most supercomputing centers use UNIX based machines, most genomic centers use them and most computational research centers use them. The presence of this section is not meant as a recommendation for UNIX. Indeed it has a steep learning curve and if you want a quick answer to one question it is certainly not worth the time. But UNIX is well suited to handling diverse and unanticipated jobs.

Unlike WINDOWS computer systems, the UNIX operating system is *in your face* and the philosophy of using the computer is quite different. In WINDOWS the operating system (from a user's standpoint) is merely a platform upon which software runs (and often the operating system gets in the way). Although APPLE machines use a UNIX operating system, again, the interface has been designed to hide this from the user. Even, most LINUX operating systems (LINUX being a type of UNIX) have slick GUI (graphical user interfaces) systems that do not necessitate knowledge of the underlying operating system.

If you have a graphical interface to the computer, you can interact with a UNIX computer in much the same way as APPLE or WINDOWS. In UNIX this interface is built upon a system called 'X' and hence you will note many pieces of software that come in multiple versions, one with an 'X' attached. For example 'maple' and 'xmaple' are text only and graphical interface versions of the same program. This graphical interface would have most of the same capabilities and, indeed, have a 'look-n-feel' that is very similar to APPLE or WINDOWS machines. UNIX machines provide many graphical interfaces, APPLE OS-X and Leopard being examples, SUN machines provide there own interface and LINUX machines provide an even large variety of graphical interfaces. Three of the most popular for LINUX machines are KDE, GNOME and IceWM.

In my opinion, however, by learning with the graphical interface you lose much of the power of UNIX because, again, in that case the interface to the operating system does it all for you. You don't learn how to do it yourself and when you want to do something more than what the interface offers — how would one do this? Is it even possible? The answer is, of course, yes and that is why I will have you painfully suffer through a command line interface.

In UNIX the operating system is designed to give you tools and make it "easy" for you to design your own tools to do any desired job. This means that you must learn something about the operating system and the many tools that are available. In addition knowing how to use just one tool is seldom sufficient to accomplish complicated tasks. Here I can provide only a brief introduction to the most important concepts and a few commands to get you started.

When you find my ramblings insufficient you can find more more information in any of a thousand books on UNIX. One I can recommend for beginners is "UNIX for the Impatient" by P.W. Abrhams and B.R. Larson, 1992/1997 (Addison Wesley). There are also many introductory packages available 'on line'. You might wish to explore Edinburgh's UNIX help for beginners or CERN's UNIX users guide, UNIX Survival, or UNIX Resources.

UNIX based workstations are built by several companies including APPLE, IBM, Hewlett Packard, Sun Microsystems, Silicon Graphics, and others. In addition there are free versions of UNIX available for a large variety of computers. Each of these has a slightly different flavour of UNIX but the minor subset of commands that we will entertain here are constant across platforms. UNIX is an old operating system (approximately 1969) with many capabilities. Along with these capabilities are often idiosyncrasies that are historical.

2.1.1 Logging on/off

When ever attempting access to a UNIX computer the user will be prompted with a request for their user identification (userID) and their password. UNIX has, from the start, been a multiuser computer system and it uses the userID to keep individual users separate and the passwords to provide a first level of security. The prompt to sign on will usually be either a request for USERID:, LOGIN:, USERNAME:, and so on. Your password will not be echoed back to the screen as you type it for obvious security reasons. Passwords should be eight or more characters long, should include some numbers or symbols, and should never be a word that is found in a search-able dictionary or database.

Upon successful access, the computer may display what kind of computer you are on, it may display when and where you last logged in from, and usually will check if you have any new mail. It will then present you with a prompt and await commands. The prompt is customizable and can include such things as the computer's name, command numbers, date, and so on.

To exit the computer simply type `exit` from the prompt. Again this is customizable to be anything you desire.

2.1.2 UNIX File System

File Structure:

Files are organized hierarchically. At the base of the file system is a root directory simply referenced by '/'. Underneath this file will be other files that can be of several different types. Under each file that is identified as a "directory", other subdirectories are possible. Different levels of directories, subdirectories, sub-subdirectories, etc. are separated by a '/'. The organization of files on a UNIX system is somewhat standardized but each vendor will do it with their own unique variations. On a typical LINUX machine the top directory contains the files ...

`boot`	
`etc`	
`lib`	
`misc`	
`net`	
`proc`	
`sbin`	
`usr`	- under this subdirectory system-wide programs unique to each installation are normally placed
`bin`	
`dev`	
`home`	- where your personal files will be located
`lost+found`	
`mnt`	- traditionally where filesystems for diskettes, cdroms, flash disks etc. will be placed
`opt`	
`root`	
`tmp`	- files needed momentarily by the system or programs
`var`	

All of these files are subdirectories (often also called folders). Most of these subdirectories contain files that are used by the operating system and most of these are files that you should never have to be concerned with. The user directories (where you can put your files) are generally placed under the subdirectory `/home`. So your home directory and where you will automatically be placed when you enter the machine will be `/home/yourid`. This is the standard location for all LINUX computers. On other machines it may be somewhere else. For example, on a Silicon Graphics computer home directories are usually `/usr/people/yourid`.

The other important directory that you should know about is `/usr/local`. This is the location where many files unique to a particular machine are normally placed. For example, in this location my machine has special files to do sequence analysis, phylogenetic analysis, etc. that are useful to every user. So rather than locating them in just one user's files, i.e. `/home/yourid`, they are stored in a central location that all can access — `/usr/local`. You may want to look there to see some of the programs installed on your local machine. The binaries (the actual executable files) for many of these programs are often stored in `/usr/local/bin`.

File Names:

Information is stored in separate files each with unique identification names. Names and extensions are arbitrary and have no reasonable limit to length or characters. The extension may be used to indicate the type of information contained in the file. Although not a requirement, a fortran file will generally have an '.f' extension, a pascal file will have a '.p' extension, and so on. Any character is permitted and even the period '.' is just another character in most file names and is not treated in any special way. Hence a filename such as "test.dat.obj" is quite acceptable. You can have a blank space as part of a filename as in "test dat obj" but this becomes quite confusing for some programs (and people) and hence is not a recommended practice.

The full name of a file will be something like `/usr/local/test.f` Note that a forward slash is used to designate different subdirectories. Here `usr` is a primary (root or top) level subdirectory, `local` is a subdirectory within this and `test.f` would be a file in the `local` subdirectory. There are not different version numbers of a file and if a backup copy is desired it must have a different name.

UNIX operating systems have always been case sensitive. The file `/usr/local/Test.f` is therefore a different file from that listed above. The difference is that a 'T' and a 't' are treated as entirely different characters. This applies not only to file names but also to commands. In general, the default will be lower case. However, if some book or example indicates that one or more letter is upper case, then this case must be copied exactly to achieve the desired result.

File Types:

There are several types of files. The most common types are textual, binary, directory, and symbolically linked files. The

latter are files that simply point to other files. There are other types but these are the ones most commonly encountered.

Paths:

Individual files can be specified in several different ways. An absolute name can be given such as `/usr/bin/ls`. Or a relative address could be used such as `../../ls` which means to go up two subdirectories levels and find a file/program called `ls`. If you simply types `ls` the computer will search for a file/program called `ls` in the current directory and if not found, will search through a specified set of directories (you can change this list).

Your Files:

You can create files in your own home directory, but you (usually) cannot create files in another persons home directory. This is dependent on the extent of the privileges you have been given. A 'long' listing of a file's characteristics would look something like ...

```
-rwxr-xr-x    1 brian    user         133 Sep 13  1996 fnd
-rw-r--r--    1 brian    user        2564 Dec 23 15:04 fumarate.pdb
drwxr-xr-x    4 brian    user        2560 Nov 26 13:55 gde96
-rw-r--r--    1 brian    user       13181 Sep 15  1996 genodb.html
drwxr-xr-x    2 brian    user         512 Nov 12 14:14 genome.sites
-rw-------    1 brian    user        3797 Jul 15  1996 hummingbird.tech
```

The first letter (here either `d` or `-`) indicates the type of file (here subdirectories or textual/binary), the next three letters give the permissions that the file's owner has to read, write or execute that file (in the case of a directory, execute is the ability to enter or view the subdirectory). The next three letters give the read, write execute permissions of anyone in the same user group, and the next three letters given the permissions of anyone on the computer. So in this example only the owner is given permission to change these files, but anybody can read/enter all files/directories except for the last one. The next number gives the number of links this file has. This is followed by the owner of these files, `brian` and the group to which this user belongs, `user`. The size of the files, the date of creation and the name of the file is shown.

Historically UNIX has been a very open system and hence the default permissions are such that any user can usually read the files of all other users. In the past this has created some security problems but these particular problems are comparatively easy to fix. You have the ability to change this default behaviour or to change the permissions of any file.

What's in a name:

Since I recommned learning the command line, it might terrify people that they are now expected to have to type out in full (and since these are computers, to type without error) a long file name that might be full of jargon. This is not necessary. Filename completion is a feature that most programs offer. For the command line this is a feature that is accomplished by a program called the "shell" (to be further discussed below). The shell on most UNIX machines usually sets up the 'tab' key to try complete the filename and if not unique, to finish it as far as possible and then present the remaining possibilities. Using this feature it is seldom necessary to type more than 2 – 3 characters of the filename.

In addition there are other ways to play with file names. An asterisk is used to match a sequence of zero or more characters. So a filename of the form

"a*.b"

will match all filenames that begin with 'a' and end with '.b'. A '?' denotes a single character but not two characters, nor zero characters. A collection or a range of characters can be matched by enclosing the range in square brackets. So a filename of the form

"numbered_files.[1-9]"

will match the files "numbered_files.1, numbered_files.2, ..., numbered_files.9". The construct '[fa2F-Z]' would match the listed characters, 'f','a','2', as well as any character from 'F' to 'Z' (remember case sensitive). Or, if you have a contrary personality, the construct '[^fa2F-Z]' would match anything but this set of characters. Finally a selection of names can be obtained by enclosing the selection in curly parentheses. The construct

'afile.{log,aux}'

will match both 'afile.log' and 'afile.aux'.

Combining all of these conventions allows rapid and flexible referencing of filenames. And as usual with UNIX, if you don't like how it is done, you can define the system to work in a different way.

2.1.3 Commands

Commands have a general structure that consists of the command name, followed by arguments. If the argument begins with a '-' it is called a flag and is used to modify the behaviour of the command. Most commonly used commands are placed in subdirectories that are searched by default and hence their complete location does not have to be specified. A typical program is run by typing

```
cmd -flag argument
```

The name of the program is `cmd` and must be somewhere that the system normally looks for program names or alternatively the path must be specified. The flags (which may or may not be present) will alter the behaviour of the program while the arguments (which may or may not be present) might include files of information to be read from or written to by the program. A few of the most basic, general commands follow.

The top ten commands	
ls	list file names
pwd	show present working directory
cd	change directory
mv	move/rename a file
rm	remove a file
cp	copy a file
cat	show file content
more	show partial file contents
mkdir/rmdir	make/remove directory
man	show manual pages

ls - list files:

This command will give a listing of files in the current directory or in the directory supplied as an argument. As mentioned abovie, to find particular files an asterisk acts as a wild card. A

```
ls a*.f
```

will list all files that begin with `'a'` and end with `'.f'`. A command such as

```
ls -l
```

will give a 'long' listing (such as was shown above),

```
ls -t
```

will sort the listing according to date and so on. There are many other flags and the flags can be combined to achieve many different responses from the same command.

mkdir/rmdir - make/remove directories:

These commands will create/delete a subdirectory with a name supplied as an argument. Only empty subdirectories can be deleted by the default command.

pwd - show present working directory:

This will print out your current location in the hierarchical file system.

cd - change directory:

Change to a subdirectory given as an argument. If no argument is given it will change to your 'home' directory. A command

```
cd ..
```

will move up one directory. A tilde is a useful character to identify home directories. The command

```
cd ~brian
```

will take you to `brian`'s home directory while the command

```
more ~/filename
```

will view the file `filename` in your own home directory.

mv/rm - move/remove files:

These commands can be used to move the location of a file. E.g.

```
mv fnd ..
```

will delete the file `fnd` from the current directory and place it one level higher. This command can also be used to rename files -

```
mv fnd dnf
```

will delete `fnd` and create `dnf`. The command

```
rm fnd
```

will simply delete the file. Be very careful with the use of asterisks and the `rm` command,

```
rm *
```

will delete all files!

cp - copy files:

This command will copy files. It requires two arguments and the first named file is copied to the second file. If the second file is a directory the file is copied to the named directory with a same filename. E.g.

```
cp fnd dnf
```

will create the file `dnf` as a copy of `fnd` but

```
cp fnd gde96/
```

will create a second copy of `fnd` in the subdirectory `gde96`.

cat - concatenate files:

In its simplest form this command will print the contents of a file to the screen. If no argument is given it will wait to accept any input that is typed from the keyboard (terminated by a <ctrl>d) and then print this out to the screen. This command is particularly useful when combined with redirection (see below).

more - print a file one screenful at a time:

This command will view the contents of a file supplied as an argument. The screen can be stepped through a file by typing

the space bar. A single line is advanced by an 'enter' key or n lines by typing the number n followed by the space bar. For simple text files, the 'b' key will move backwards a screen.

lpr - print:

The `lpr` command will send a file (argument) to a printer. In general most UNIX machines are set up for postscript but individual printers can be set to accept other types of input. Indeed many modern printers will switch 'on the fly' to match the input it is receiving. Postscript is a graphics language that describes the structure of a figures (e.g. a circle of width x) rather than individual points (e.g. actual bit mapped points). In this way it is independent of the resolution of any viewer, simply providing instructions on how to display a figure at the viewer's maximum resolution. You can recognize if a file is postscript by either the filename extension (`*.ps`, `*.eps` or rarely, `*.epsi`) or by its contents. A postscript file will usually begin with something like

```
%!PS-Adobe-2.0
%%Creator: WiX PSCRIPT
%%Title: ramermap1.cdr FROM CorelDRAW!
statusdict begin 0 setjobtimeout end
statusdict begin statusdict /jobname (ramermap1.cdr FROM CorelDRAW!) put end
{}stopped pop
{statusdict /lettertray get exec
       ....
```

If the file is not in postscript and the default printer on your system is an old printer and expects postscript then you must translate the file first. A common (and free) program to do this is `a2ps`. This program takes a file (supplied as an argument) and changes it to postscript (along with many other abilities), and then automatically pipes it to `lpr`.

For both `lpr` and `a2ps` (and many other programs) the -P*printername* flag will direct the output to the particular printer chosen. For example, the command

```
a2ps -Pps filename
```

will translate the file `filename` to postscript and pipe the output to the printer named `ps`.

2.1.4 Help

All of the above commands have many other abilities. To find out about these abilities there are manual pages stored on the computer. Typing

```
man cat
```

will generate a manual page that describes this command and then passes this page to the `more` viewer.

If you have a graphic interface you can also run `xman` which has more capabilities. There is a move afoot to replace the man programs with a similar but more advanced program called `info` (but I still prefer the old system). You will also often find files under the directory `/usr/doc` or `/usr/share/doc` (along with the directory `/usr/share/doc/HOWTO` which is particularly useful for beginners).

In addition you can search an index of manual pages with `man -k word`. All manual pages that are considered relevant to `word` will be listed. If you want to know more about the `man` command, type

```
man man
```

(of course).

2.1.5 Redirection

To UNIX, the keyboard and the screen are just different types of input/output streams. If desired you can redefine these. For example, you can redirect output of a command such that it is not put onto the screen but rather put into a file. For example the command

```
ls -aF > myfiles
```

will run the command

```
ls -aF
```

Input to the command ends at this point and the part

```
> myfiles
```

instructs the computer to put the output of the command into a new file to be called `myfiles`. So

```
cat fnd > dnf
```

is equivalent (well, . . . close enough) to

```
cp fnd dnf
```

In general,

- `>` will "send output to . . . "
- `>>` will "append output onto the end of . . . "
- `<` will "take input from . . . "

You can also specify a "pipe" symbolized by '`|`' which will take the output of one command and use it as input for another command. It is kind of like a

```
> <
```

command. For example you could enter the command

```
ls /usr/lib | more
```

This will take the file listing of subdirectory `/usr/lib` and give that information to the `more` command.

2.1.6 Shells

Shells are a command interpreter that will be run on all UNIX computers. You can think of the shell as a layer of program through which all of your commands are passed before being processed. Again there are many different shells. The popular ones are the 'sh' Bourne shell, 'bash' Bourne again shell, 'ksh' Korn shell, 'csh' C shell, and the 'tcsh' shell. The latter is the default shell run on the computer you will be using (though the bash shell is generally recommended).

The `tcsh` has several nice capabilities (many shared by the other shells). One of these is filename completion. If you type the beginning of a filename and then type 'tab', the computer will finish this filename. If the request is ambiguous the computer will finish the filename as far as possible and then beep. Typing `<ctrl>d` will display matching filenames up to that point.

This shell also keeps a numbered history of your last commands. You can step through them using the 'up'/'down' arrow keys. The commands can then be repeated or edited. For example, if you type

```
ls
 cat fnd
```

and then type the 'up' arrow twice. This will return you to the `ls` command. Alternately, you could rerun the command by typing just

```
!l
```

An exclamation mark followed by a string will repeat the last command beginning with that string. An exclamation mark followed by a number will rerun that numbered command. The command `history` will give a numbered list of your past commands.

This shell also permits the creation of aliases. Aliases can be set up as follows, type

```
alias dir `ls -aF`
```

Thereafter, typing `dir` will run

```
ls -aF
```

(The 'a' flag shows hidden files and the 'F' flag adds a '/' to end of a directory, a '*' to the end of a binary, a '@' to the end of a link. Be careful as these are not actually part of the file name). Aliases can be bypassed by preceding them with a backslash. Thus

```
\dir
```

will return `dir` to its original definition and ignore the alias (in our case `dir` is not a defined command and you will get an error message).

2.1.7 Special 'hidden' files

Files that begin with a period are called hidden files and are not shown by a default `ls` command. You can 'see' them with an

```
ls -a
```

command.

Two of these files are `.cshrc` and `.login`. These files are read (and the commands inside executed) every time you start up a csh (or tcsh) shell and every time you login to get onto the computer. The file `.cshrc` contains many aliases and you can edit this file and add your own. Your default path to search for files and commands can also be defined in this file.

Many programs may define a `.xxxrc` file. They use this file to read and store variables that will be used in the programme.

2.1.8 Background Processes

UNIX users are notoriously impatient. If the computer is taking too long to finish a job, it can be put into "the background". This means that the computer will work on this process and at the same time, present to you another prompt for your next command. The way to put a job into the background is to add a '&' at the end of the line. A job number will be supplied to you and then the computer will start working on that process. Any output from this command will be sent to the screen or a file as appropriate but it cannot accept interactive input in this state. Thus the command

```
ls &
```

will run `ls` in the background and present you with another prompt. (But hopefully on my machines the `ls` should be done before you get a chance to type in anything else).

Another way to do this, particularly if interactive commands are required only at the beginning of a program, is to type `<ctrl>z` when the process has started its work. This will suspend the job (the job is not killed but nor is it active). To restart the job type "fg" (mnemonic foreground). To put the job into the background type "bg" (mnemonic background).

To check on the jobs that you have running use a `ps` command. This will list the processes that the computer is currently working on. To cancel a job use

```
kill pid
```

where "pid" is the number associated with the job according to the `ps` command. Alternatively

```
kill %n
```

where 'n' is the number of the job given to you when you typed '&'. Finally, to kill a program that is currently executing (assuming it will still accept input from the keyboard), enter `<ctrl>c`.

2.1.9 Utilities

UNIX has many standard utilities that are very useful but I can only talk about a few here. Perhaps the most used is the search utility that will find text in a file/files. There are a family of "grep" commands that perform these searches. The command

```
grep -i Frank /usr/*/address.bok
```

This command will search all subdirectories under `/usr` that have a file named `address.bok`. It searches inside these files for the text `Frank`. The flag `-i` causes the search to be done in a case insensitive fashion. As an example, to see only the process that the machine is running for you rather than all processes, type

```
ps | grep yourid
```

Depending on what you wish to do, there are also `egrep` and `fgrep` variants of this utility. Some other commonly used utilities are `sort, cut, paste, diff,` and `tr`. For information on these see the `man` pages.

2.1.10 Editors

There are many editors available both for free and commercially. If you have used `pine`, you have used the `pico` editor. The most common and ubiquitous editor is `EMACS`. `EMACS` is available on most UNIX computers but, in the past, it has been rather picky about the terminals it will talk with.

An older, more basic and works from anything editor is called `vi`. This editor was designed to work without the aid of a mouse and to permit easy mapping of keyboards to accommodate multiple hardware manufacturers.

This editor is invoked by typing

```
vi filename
```

Again, if the file does not exist then it will be created by this command. There are two modes to the basic editor – command mode and insert mode. In the former mode, everything typed from the keyboard is treated as a command while in the latter mode, everything typed from the keyboard is added to the file. To change from insert mode to command mode use the 'escape' key. There are several ways to change from command mode to insert mode. To insert text after the cursor hit the 'i' key while in command mode. To append text to the end of a line use 'a' while in command mode. Use the 'x' key

Table 2.1: A few vi Editor commands

ESC	return to command mode.		
i	change to insert mode.	a	append to the right of cursor.
:w	save file	ZZ	save the file and exit.
:q	quit	:q!	abort edit.
w	move right a word.	b	move left a word.
H	move to top of screen	M	move to middle of screen
L	move to bottom of screen	^F	scroll one screen forward
^B	scroll one screen backward		
cw	change word	cc	change line
~	change case	C	change rest of line
s	substitute character under cursor		
u	undo last command.	U	undo all changes to line
x	delete character	dw	delete word
dd	delete line	:5,10d	delete lines 5-10
:set nu	show line numbers.	:set nonu	hide line numbers.
11yy	copy 11 lines into buffer	11dd	delete 11 lines into buffer
p/P	put buffer below/above line	:1,2 co 3	copy lines 1,2 to after line 3
:1,2 m 6	move lines 1,2 to after line 6		
G	go to last line.	11G	go to line 11.
/str	search for str.	?str	search backwards for str
n	find next occurrence		
:g/search/s//replace/g	find and replace		
:g/search/s//replace/gc	same but consult		
v	locally defined to reformat paragraphs (it is mapped to !}fmt)		

to delete characters under the cursor. In command mode 'dd' will delete entire lines, so do not idly type keys while in command mode. The arrow keys can be used to move around (albeit slowly). This editor has many commands and many capabilities (see Table 2.1 or see a book on UNIX for more). To exit the editor, move to command mode by hitting 'escape' and then type 'ZZ' (note upper case!). Your work is automatically saved but a backup of the state of the old file is not generally made.

On my computer `vi` is again aliased and in reality typing `vi` will invoke `vim` instead. This is a modernized version of `vi` which is actively being developed (2008). This project has included mouse support if you have proper terminal definitions for mouse standards (e.g. an `xterm` interface). It also has a graphic interface started by `gv` (another alias, actually `gvim`). This update includes many features, the best being easy customization and simple programming abilities. To find out more check out the vim web site, type

```
:help topic
```

inside the editor (note the preceding colon), or examine the free online documentation "vimbook-OPL.pdf" (follow the link from http://www.vim.org/docs.php) or the book Hacking Vim (a book that support orphans in Uganda as part of the vim project), or the commercial book Learning the vi and Vim editors.

2.2 Exchange among computers

2.2.1 ssh

The programs `ssh` and `scp` are programs that you should use in preference to `telnet` and `ftp`. These programs are replacements for the older commands `rsh` and `rcp`, where the more logical 'r' stood for remote (hence to open a shell on a remote computer – rsh, or to do a remote copy – rcp). The change of the 'r' for the 's' stands for secure and the difference between `rsh` and `ssh` is that the information is encrypted before it is sent across the internet (including encryption of any transmitted username and password) and then de-encrypted 'on the fly' at the remote computer location. The encryption is different each time a different connection is made and is difficult (but not impossible) to crack. To use `ssh` simply type

```
ssh remotehost
```

and you are off. You might see some information about the nature of the encryption, about exchange of keys and so on. For `scp` the commands are the same as `cp` except that you can use a ':' to separate file names from machine names. For example,

```
scp george@california.edu:stuff/filename1 frank@newyork.edu:filename2
```

will copy a file named `filename1` in subdirectory `stuff` under the home directory (default) of user `george` on a machine in California to user `frank` on a machine in New York even if you are a third user sitting on a machine in Canada (of course you will have to have passwords to all three accounts; george's, frank's and the password for the machine in Canada). See the `man` pages for further information on these protocols.

There are other programs that can be saved for later.

2.2.2 Mail

I have been stressing in this course the utility of a command line interface to UNIX. There are of course, therefore, character based interfaces to e-mail. The two most popular character based interfaces are `pine` and `mutt`. Each are easy to use and more importantly you can, with ease, include them in programs that you write (so for example, mail to Frank the results of the program and send the ancillary data generated to Susan; or send off formated emails to specific people based on the program results). These programs have some simple properties that make them very easy to use. As just one example to send a mail message with hundreds of attachments (and annoy your friends) simply enter the command

```
pine user@remote.machine -attachlist directory/*
```

This command will send an email to a particular user on a remote machine and will attach to this email all of the files in the directory 'directory'.

2.3 Scripts-Languages

Even if UNIX provides a wide variety of efficient tools to accomplish generalized basic tasks these tools cannot solve all problems. Hence a bioinformatician must learn a programming and/or a scripting language.

All UNIX systems come with a variety of scripting languages and a variety of programming languages (usually pre-installed). The simplest of these is usually the scripting language of the shell itself. Beginning a file with the line

```
#!/bin/sh
```

(or csh, or bash, or zsh, etc.) and changing its permissions to be executable (with the command, 'chmod u+x file'), will execute each of the commands that follow in the file according to that shell (the experts recommend scripting in the sh shell or a modern shell rather than csh). If the line is missing, the current shell will be used. Any command and any series of commands can be put in these files. More capabilities are offered by the sed script editor and more still by the awk programming language.

Most systems will also include more extensive programming languages. Since UNIX is built upon C, almost all machines include the C programming language (and more recently, the C++ programming language). Other popular computer programming languages include java, perl, PHP and python. Some computers may include fortran (f77), pascal (pc), tcl/tk and there are many others. I would recommend that students learn at least one scripting language for quick and simple tasks and learn at least one compiled language for more computationally intensive tasks. In my group we currently use perl for rapid scripts and we use C for intensive things.

For each of these you can obtain limited information from the man pages. But to actually learn how to use them, you should find some books or examine instructional web pages (indeed the first prize ever offered for a web-based courses went to a site teaching C++ at MIT).

Detailed instructions on learning a computer language are beyond the scope of these notes. I would recommend, Developing Bioinformatics Computer Skills by C.Gibas and P.Jambeck (2001) and Beginning Perl for Bioinformatics by J.Tisdall (2001).

With respect to bioinformatics and sequence analysis there are important resources of which you should be aware. There are libraries of subroutines and objects (bits of computer language code) that you can incorporate into your own programs. Many of these libraries are publically and freely available for all to use. An extremely useful collection of code, the perl library can be found at www.bioperl.org/wiki/Main_Page. Java libraries can be found at biojava.org/wiki/Main_Page, a C++ library can be found at http://kimura.univ-montp2.fr/BioPP and a suite of programs based on this library called Bio++ can be found at http://home.gna.org/bppsuite/, the Phylogenetic Analysis Library (PAL) can be found at www.cebl.auckland.ac.nz/pal-project/; an effort being led by Dr. A. Drummond and Dr. K. Strimmer, and a collection of algorithm libraries for bioinformatics (AliBio) designed to be fast and efficient can be found at http://www.bioinformatics.org/project/?group_id=173.

2.4 Obtaining LINUX

If you have an APPLE computer, then you are already using UNIX. The underlying operating system is based on a version of UNIX. You can use the 'terminal' command to get access to the command line.

If you have a WINDOWS computer then you can get access to a remote UNIX computer if you have an "ssh" program on your computer. There are free versions of these programs available for download and commercial versions.

If you would like to try out a more complete UNIX experience without giving up WINDOWS, you can download LINUX operating systems for free and install them on your computer with the option of leaving the computer dual-bootable. What

this means, is that when the computer is started, it will ask whether you wish to begin WINDOWS or LINUX and then launch whichever operating system is chosen. This type of an installation will automatically partion your drive and hence, as always, it is a good idea to backup your files.

Free versions of LINUX are available at . . . ,

Ubuntu	www.ubuntu.com
Debian	www.debian.org
SUSE	www.novell.com/linux/
Mandriva	www.mandriva.com

(these are different flavours of LINUX by different groups or companies; there are others in addition to these). The companies also offer support but this usually is no longer free.

If you are unsure or would just like to see what UNIX things are like, try a KNOPPIX or a "live" distribution. This is a form of LINUX that runs from a DVD or from a memory stick. To start KNOPPIX or a live distribution simply insert the DVD or memory stick into your computer and restart the machine. These systems do not alter the hard drive of the computer and simply run the entire operating system off the removeable device (with the result that it can be a little slow). To test drive this system make sure that when you download the operating system that you burn a 'bootable' image of the DVD and make sure to set your computer to boot from the device. Simple instructions to do both of these are on the KNOPPIX web site at www.knoppix.org or at the sites listed above.

I am happy to recommend a DVD, memory stick or full installation of BioLinux. This is a customized LINUX distribution that comes customized with hundreds of bioinformatic tools all pre-installed. BioLinux can be obtained from http://envgen.nox.ac.uk/biolinux.html out of NERC England.