Introduction to

Bio-Linux

For Bio-Linux 7
December 2013

NERC
Environmental
Bioinformatics
Centre

Website: http://nebc.nerc.ac.uk/tools/bio-linux

Email: helpdesk@nebc.nerc.ac.uk

Table of Contents

Part One: Introduction to the Bio-Linux 7 System

Logging in and exploring the Bio-Linux desktop

You can log into your Bio-Linux machine locally or remotely, on a fully installed system or on a system running Live from a USB memory stick or a DVD.

These course notes are written from the perspective of someone running the Live version of the system – that is, having booted the PC directly from a USB memory stick. The main differences for people working on an installed system will be the name of the account you are logged into and what privileges that particular user account has. The user of the Live system always has full administrative privileges.

Please refer to our on-line document about various ways you can access Bio-Linux systems:

http://nebc.nerc.ac.uk/tools/bio-linux/other-bl-docs/installoptionsleaflet

If you are booting the machine from a DVD or a USB memory stick to run the system live, then choose

Option 1: Try Bio-Linux 7

After the system has started up, you will see the Bio-Linux desktop (Figure 1).

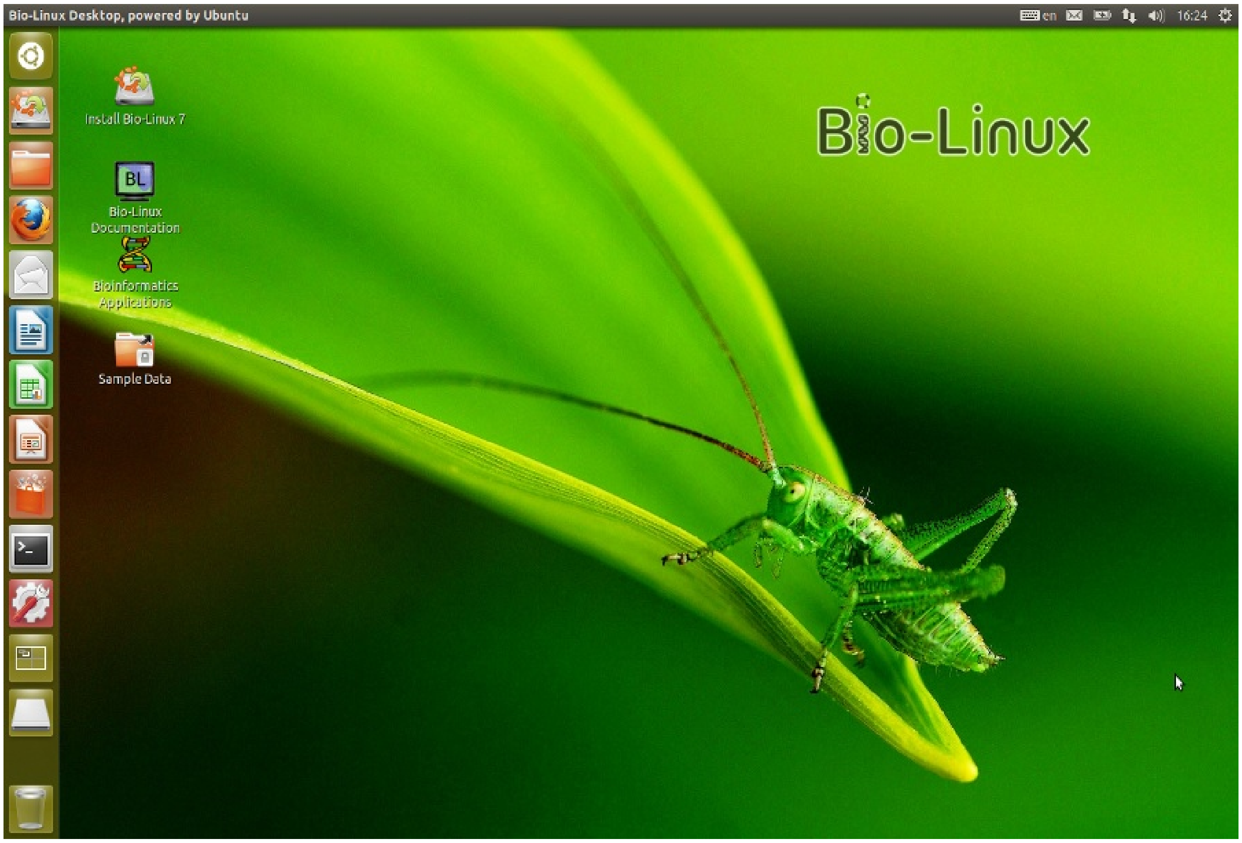


Figure 1: A view of the Bio-Linux 7 desktop

There are four icons on the desktop

- **Install Bio-Linux 7** On the Live System only – click this icon to start the Bio-Linux installer
- **Bio-Linux Documentation** Opens a menu of links as follows:
  - **NEBC Homepage** Opens the NEBC home page in a web browser
  - **User Guide** Opens the Bio-Linux Userguide – a basic introduction to system admin
  - **Introductory Tutorial** Opens the folder of Introductory Bio-Linux tutorials and data files
  - **Bioinformatics Docs** Shows the NEBC Bio-Linux Bioinformatics Documentation System
- **Bioinformatics Applications** Opens window of icons to some of the main bioinf. programs (see below)
- **Sample Data** Provides access to much sample data to help you in trying out new software

On the left of the screen you will see the **Dash,** which is used to launch and organize applications. The dash is populated by a column of large button icons. The button at the top with the Ubuntu logo brings up the main Dash panel to find files and applications (see below). The other icons are, by default, from the top:

1. Open the application and media panel
2. Install Bio-Linux
3. Open your home folder
4. Launch Firefox web browser
5. Launch Evolution mail reader
6. LibreOffice Writer word processor
7. LibreOffice Calc spreadsheet
8. LibreOffice Impress presentation editor
9. Ubuntu Software Centre (find and install apps)
10. Shell Terminal
11. System Settings
12. Virtual Desktop Switcher
13. Access USB removable media
14. Rubbish Bin (deleted files area)

On the top of the screen you will see the menu and panel bar (Figure 2).

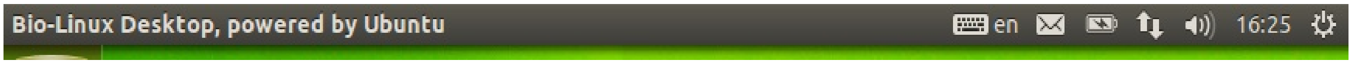


Figure 2: The menu and panel bar, found at the top of the screen.

If you open an application window, the name of the active application will appear in the left portion of this bar. If you move the mouse over it, a context menu for the active window will appear. The right portion of the bar has a panel of icons to control some system settings.

From left to right, the things you see in the top right panel area are:

1. Keyboard selector (defaults to UK keyboard)
2. The envelope icon allows you to set up Chat or Mail on the system.
3. Battery monitor (on laptops only)
4. Network monitor and setup (double arrows icon indicates a connection is active)
5. Audio volume control
6. Wall clock (click for a calendar)
7. System menu (includes access to system settings and options to lock screen, switch user, shut down, etc.)

Running applications

There are several ways to start applications on Bio-Linux. The most obvious is to double-click the **Bioinformatics Applications** icon on the desktop. This brings up links to some of the main bioinformatics applications, but by no means all of them. Most applications need to be accessed from the terminal, as detailed at length in this tutorial.

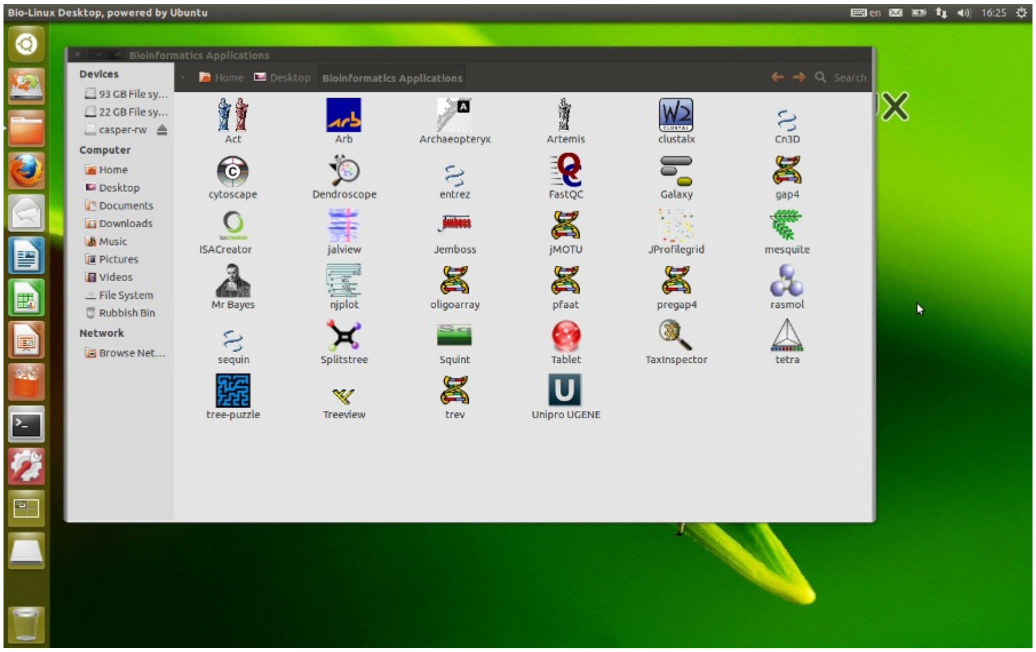


Figure 3: The Bioinformatics Applications folder

Clicking the big button at the top of the Dash opens a panel where you can search for applications and files on the system. This includes bioinformatics tools and any other applications you have installed. Start typing either the application name or a keyword, or select the icons along the bottom of the panel for a different view.

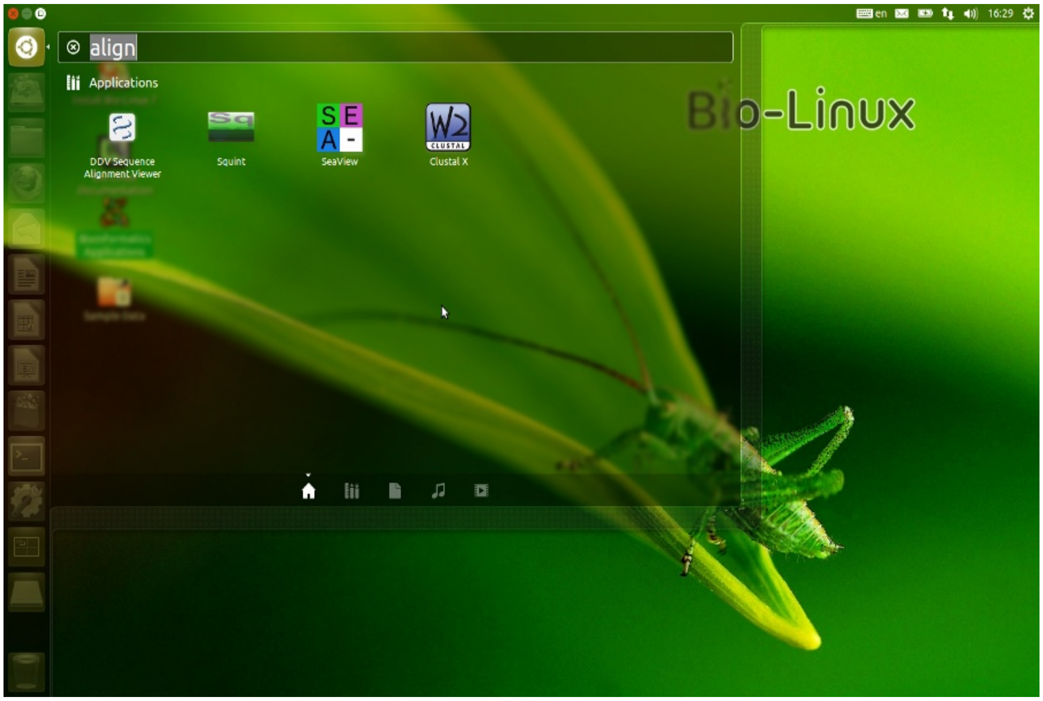


Figure 4: Searching for applications in the Dash

Finding files and drives

The orange folder icon near the top of the Dash takes you directly to your Home folder.

●

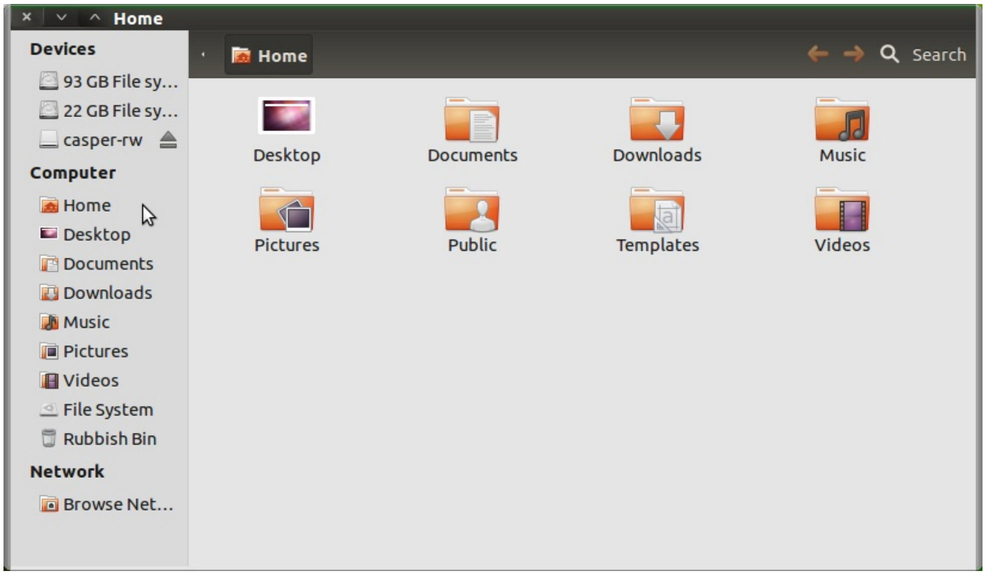


Figure 5: Your home folder

Your personal Desktop, and folders in your Home area called Documents, Pictures, Videos, etc. are listed. You can use these or else create your own folders as you wish.

The file browser provides convenient shortcuts to these directories in the left pane, even if you are viewing another folder in the main panel.

Devices recognized by your system such as the disk drives, CD/DVD devices, USB sticks, etc. are listed at the top of the left pane. Removable media can be ejected by clicking the icon next to the device name.

Networks resources can be accessed through the **Browse Network** icon. This includes Windows network shares and files on your other Bio-Linux machines, that can be accessed via the SFTP protocol. Browsing regular FTP servers is also supported.

Note: The Dash also has a file and media finder, as seen on the previous page, selected by clicking the Ubuntu button at the top left to bring up the Dash console and then selecting one of the little white icons from along the bottom of the window.

Setting things up

The **System settings icon** allows you to customise and administer your system (Figure 6) in various ways.

The **Personal** area is used for customising a variety of attributes relating to your personal preferences.

The **Hardware** and **System** areas allow you to do things such as configuring hardware drivers, changing firewall settings, administering users and groups, and managing the packages on your system.

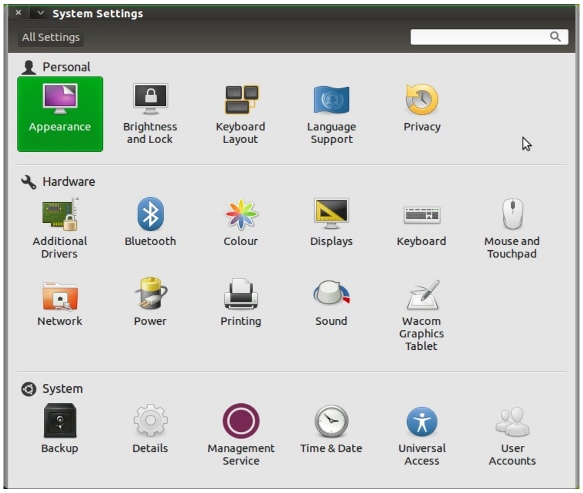


Figure 6: The System Settings Window

Other features - Virtual Desktops etc.

The icon that looks like this: allows you to switch “virtual desktops”. Unlike Windows, Linux by default gives you access to multiple desktop areas. This allows you to have windows open for different things in different virtual desktops. For example, if you were working on writing an article, you could have programs relevant to that work open and visible via one of these desktops. Meanwhile, you could have programs related to sequence analysis open on another desktop, and so on. This is a great tool for keeping things organised during your working day. Clicking the icon will zoom out to show an overview of all desktops. You can also switch quickly by holding down Ctrl+Alt and tapping the arrow keys on the keyboard.

The Deleted Items Folder icon (also commonly referred to as a Recycling Bin or Trashcan) is the bottom icon the Dash. This is where files deleted using graphical tools usually end up. This gives you a chance to salvage them if you deleted them by mistake. Deleting files on the system is covered in more detail in the *Removing Files and Directories* section of this tutorial.

Exercise 1-1

a) Exploring the desktop

Take some time to explore the desktop. Look at the options under each of the icons covered in the previous section, and try the various options in the Dash console. Try clicking the icons on the desktop. Also try using the right and middle mouse buttons when the mouse pointer is over the icons in the Dash and explore the menus presented to you.

Try going to a different virtual desktop and starting up some windows/applications there. Try dragging windows of one desktop area and onto another.

continued over...

(Exercise 1-1, continued)

b) Unpacking the example files for this tutorial

The sample files referred to in this tutorial can be found on the system as a compressed package file. You'll need to copy and unpack them before proceeding.

Option 1 – copying the file from the tutorials folder on the system

- Double-click the **Bio-Linux Documentation** icon on the desktop
- Open the **Introductory Tutorial**
- Drag the **bioinf_files.tar.gz** file to the left and drop it over the word **Home** to copy it to your home folder.

Option 2 – as per 1, but using a command in the terminal

- Open a terminal by clicking on the Terminal icon in the Dash toolbar.
- Type the following command, then press the Enter key:

```
cp  /usr/local/bioinf/documentation/bio-linux/intro_course/bioinf_files.tar.gz   /home/live
```

Option 1 and 2 show how to copy this file either with a graphical tool or by typing a single command. The result will be the same.

The sample files can also be downloaded in a compressed package from our web server, and again you can do this either within a web browser or on the command line. The file is here:

http://nebc.nerc.ac.uk/downloads/courses/Bio-Linux/bioinf_files.tar.gz

Option 3 – downloading via the command line

The ***wget*** command knows how to fetch files from a web server, so you can do this:

```
wget  http://nebc.nerc.ac.uk/courses/Bio-Linux/bioinf_files.tar.gz
```

Option 4 – downloading via a web browser

- Open the Firefox web browser by clicking the Firefox icon on the Dash toolbar
- Type the address given above into the web browser URL address field.
- Save the download file to your home directory: /home/live.

c) Extracting the files from the compressed tarball

The file you just downloaded is referred to as a tar file or "tarball". Tar is a utility similar to Winzip; it makes package of files. The .gz extension shows that the gzip method has been used to compress the tar file.

Again, here are two equivalent options for how to unpack these files. One command line and one graphical. Choose whichever you are most comfortable with, or try both.

continued over...

(Exercise 1-1, continued)

Option 1 – extracting via the command line

- Type the following on the command line:

```
tar -xvz -f bioinf_files.tar.gz
```

This command uncompresses and unpacks the contents of the tar file into the directory you typed the command from. The letters after the hyphens are parameters of the **tar** command: **x** means "unpack/extract", the **v** means "in a verbose fashion, reporting to the screen the list of files being unpacked", the **f** means that the filename you are giving is the tar file that will be unpacked, and the **z** means that the file is compressed with **gzip** and should be uncompressed.

Option 2 – extracting via a graphical interface

- Open your **Home Folder** by clicking the orange folder icon in the Dash.

- Click the right mouse button over the bioinf_files.tar.gz file and move the cursor down to **Extract To...** (Figure 7).

- Choose to extract the file in its current location by clicking **Extract**. Note that in the dialog that appears here, your home folder will be shown by the system name **live** rather than as **Home**.

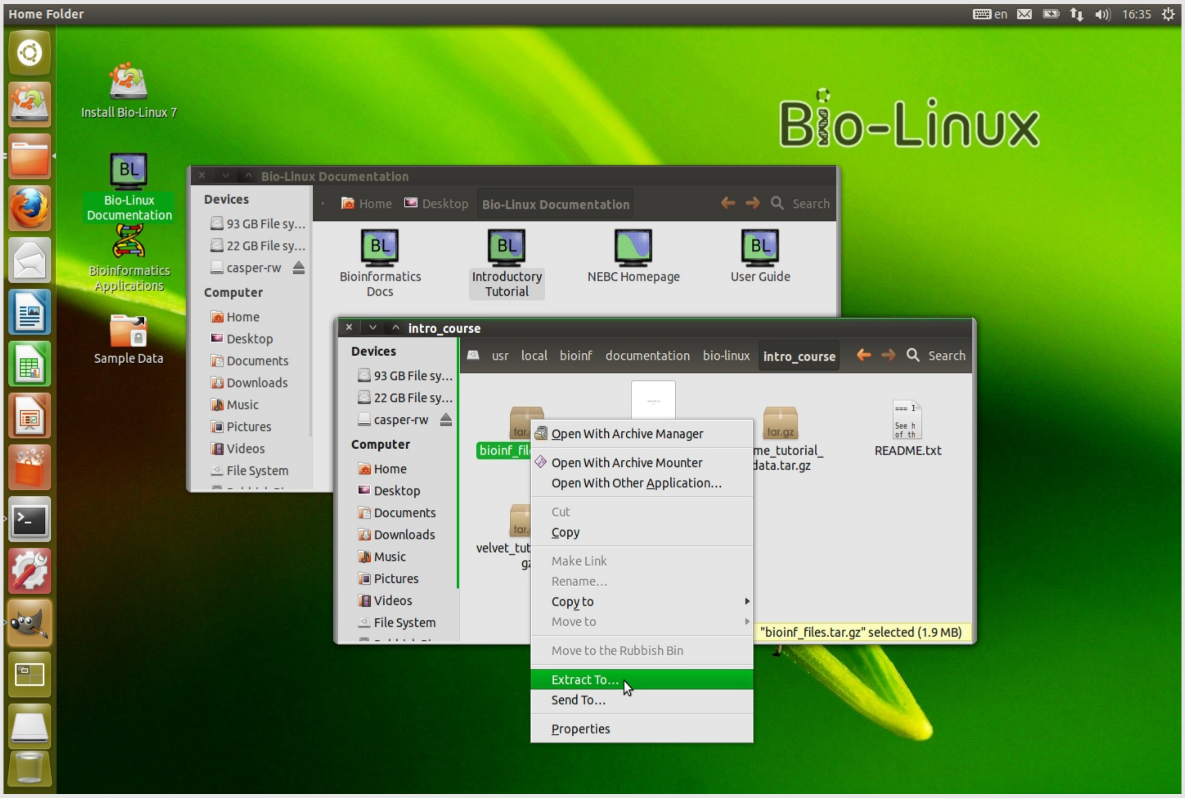


Figure 7: Extracting a .tar.gz file via the right-click menu.

continued over...

(Exercise 1-1, continued)

d) Removing the compressed tarball

The unpacked files that you will be working with in this tutorial are now in a directory called **bioinf_files**.

You can remove the compressed tar file now if you wish. Again, this can be done via the command line or using the graphical file browser. Here we just show the command line version. More details about how to remove files from the system are covered in the *Removing Files and Directories* part of this tutorial.

- Open a terminal window: click on the Terminal icon in the Dash.
- Type the following into the terminal, then press Enter:

```
rm bioinf_files.tar.gz
```

- ***Enter "y" to agree when you are asked if you wish to delete the file.***

Finding your way on the system

In Linux /Unix systems, documents are usually referred to as **files**, and file folders are referred to as **directories**.

Your Bio-Linux machine can be thought of as a huge file folder (directory), inside of which are many other file folders (directories). Inside these there are more nested file folders (directories), and so on. As in the real world where file folders can contain documents and other file folders, in Linux, directories can contain files and other directories.

Your personal Home folder is one directory within the set of directories that make up your Bio-Linux machine. In your account, you can create other directories, store data, run programs, etc.

By default on a Bio-Linux machine, you have the right to create, delete and edit files in your own Home folder, but not in other people's accounts. You can be given permission to work on files in such areas, and some information on setting file permissions is given later in this course. Your system administrator or local IT support should be able to help you with sharing files on a shared server.

You can use command line or graphical tools to explore directory areas on the machine, including your home directory.

A graphical view of your home directory is available by clicking on the orange **Home Folder** icon in the Dash toolbar (Figure 5). This opens up a window that shows the files and directories in your home directory. The full name of this folder on the system is **/home/live,** reflecting the default user name on the Live system, and you will need to use that name when working in the terminal, but the graphical file browser just shows it as **Home.**

This graphical representation of your files is similar to what you might see on a Windows PC. It is a good way to visualise the directories and documents you have, and to move around in your account. It allows many of the same actions that Windows file management systems, including copying, moving or deleting files using drag and drop or copy/cut/paste.

You can access all the areas of the file system outside your home directory through this window also. To do this, click on F**ile system** in the left hand pane to see the "root" directory of the system.

Exercise 1-2

- If you have not done so already, click on the orange **Home Folder** icon near the top of the **Dash**
- Double-click on the **bioinf_files** directory that you unpacked in Exercise 1-1, to view the contents
- Investigate the options under the file browser menus. These appear on the bar at the very top of the screen.
- Click on the ***File System*** icon in the left panel. This allows you to see the root directory – the base of the whole filesystem hierarchy.
- Find the folder called ***home*** and double click on it.
- You should see your ***Home*** directory listed. Select this to get back to your home folder. If you are on a Live system, the folder for your Home directory will be called *live*. If you are working on an installed system, you should see a folder with your username; other user folders may also listed. A lock symbol on a folder would inform you that you do not have permission to view the contents of that folder.

The Root Folder

The name of the base directory of the whole system, the one within which everything else on the system is contained, is the **root directory**. It is referred to by a single forward slash " / ".

When you opened your home directory in Exercise 1-2, you should have seen boxes similar to those in Figure 8 in the File Browser window. You need to click on the arrow to the left of **Home** to reveal the full path within the file system heirarchy.

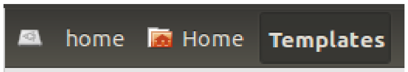


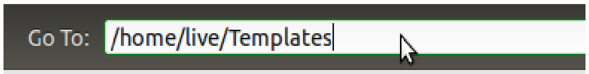
Figure 8: Location path for Templates folder in File Browser Button View.

This tells you that your personal home folder (actually called **live** but labeled as **Home**), sits within the directory called **home** (with a small **h**), that contains homes for all users. This directory **home** is under the **root** directory, represented by the tiny picture of a disk.

In other words, this information tells you where you are in system.

The location of a file or directory within the system is its **path**. If you are asked for the **full path** or **absolute path** to a file, you need to provide a complete listing of all the directories traversed on the system to get to that file. That is, you need to give the full path from the **root directory** to that file. The path is written by starting with a ***forward*** **slash** "/" then listing the names of the directories you need to traverse in the system to find that file, with each directory name separated with another **forward slash**.

To see the full path in the format most command-line programs would expect you to provide, press **Ctrl-L** while viewing a File Browser window. You should see something like this:



Go To: /home/live/Templates

Figure 9: Location in graphical file browser given in text; this is the the full path to the Templates folder in the home directory of the **live** user account.

To summarize the syntax provided in Figures 9 and 10:

/home **home** is a directory located within the root directory

/home/live **live** is a directory within the directory **home** which is within the **root** directory. This special directory will sometimes be shown as **Home**, with a capital **H**, because it is the home folder for the live user.

As another example: the **full path** to the file **capsall.fasta**, in the **bioinf_files** directory within the **home** directory of the live user:

```
/home/live/bioinf_files/capsall.fasta
```

Sometimes you can provide just the route from where you are on the system to where your file is; this is referred to as the **relative path**. For example, if you are working in your home directory, the relative path to the file mentioned above would be **bioinf_files/capsall.fasta**

Keeping things organised

Everyone knows it, but it's worth restating: if you start by creating a folder structure with meaningfully named subfolders, name your files so that the names indicate the contents (or follow some defined naming convention), and store your files in the right place, your life will be ***much, much easier!***

Using the command shell

The real power of Linux/Unix systems is the command line.

*A list of common Linux commands is provided in **Appendix D** of this document for reference.*

Many programs and facilities are available through graphical options on Linux, but ***all*** programs and facilities can be accessed by the command line, also known as the **shell**. Some tasks are easier, or more appropriately done using graphical interfaces. Equally though, other things are easier or more appropriately done using the command line. Obvious examples include when you need to work with large numbers of files or want to automate processes. First steps on the command line can be hard but the rewards are worth it (we promise!)

Access to the command line is done through a **terminal** window.

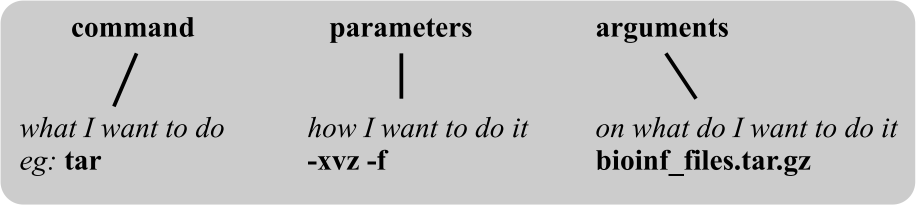
You can open a new terminal by:

- clicking the middle button on the **terminal icon** on the Dash toolbar
- or, going into an already open terminal and typing a command to open a second terminal:

```
gnome-terminal &
```

Anatomy of a Command

Linux/Unix commands usually take the form shown in Figure 11.   You've already seen a good example in Exercise 1-1 part c.



command	parameters	arguments
what I want to do	*how I want to do it*	*on what do I want to do it*
eg: **tar**	**-xvz -f**	**bioinf_files.tar.gz**

Figure 10: The Linux/Unix command line structure. Each part of a command is separated by one or more spaces.

The first word you supply on the command line is interpreted by the system as a command; that is – something the system should do or a program to be run. Items that appear after that on on the same line are separated by *spaces*. The additional input on the command line indicates to the system how the command should work. For example, what file you want the command to work on, or the format for the information that should be returned to you.

Most commands have options available that will alter the way the command functions. You make use of these options by providing the command with *parameters*, some of which will take *arguments*. Examples in the following sections should make it clear how this works. With some commands you don't need to issue any parameters or arguments. Occasionally this is because there are none available, but usually this is because the command will use default settings if nothing is specified.

If a command runs successfully, it will usually not report anything back to you, unless reporting to you was the purpose of the command. If the command does not execute properly, you will often see an error message returned. Whether or not the error is meaningful to you depends on your experience with Linux/Unix and how user-friendly the errors generated were designed to be.

Note: Items supplied on the command line separated by spaces are interpreted as individual pieces of information for the system. For this reason, a filename with a space in it will be interpreted as two filenames by default. This is addressed in more detail later in the course.

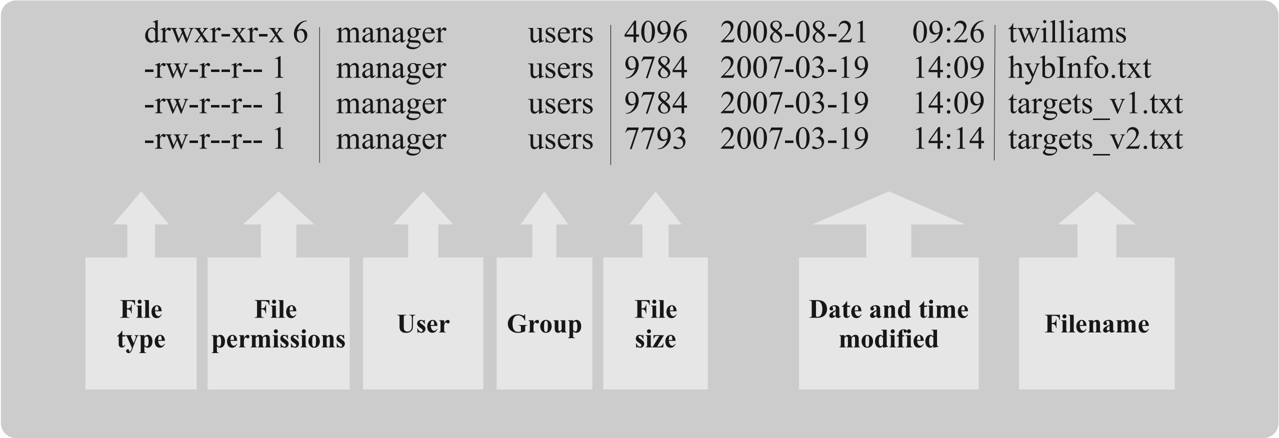
Note 2: The use of the ampersand in the previous example, **gnome-terminal &**, is explained in a few pages time. You would not put an ampersand on the end of most shell commands.

Listing files in a directory

The command **ls** lists files in a directory.

By default, the command will list the filenames of the files in your current working directory. At the moment, this is probably your home directory.

If you add a space followed by a **–l** (that is, a hyphen and a small letter L), after the **ls** command, it alters the behavior of the command: it will now list the files in your current directory, but with details about them including who owns them, what the size is, and what kind of file it is. Information about this is shown in Figure 11.



```
drwxr-xr-x 6 | manager      users | 4096  2008-08-21   09:26 | twilliams
-rw-r--r-- 1 | manager      users | 9784  2007-03-19   14:09 | hybInfo.txt
-rw-r--r-- 1 | manager      users | 9784  2007-03-19   14:09 | targets_v1.txt
-rw-r--r-- 1 | manager      users | 7793  2007-03-19   14:14 | targets_v2.txt
```

Figure 11: The detailed output of the command **ls** when run with the **-l** flag

Exercise 1-3

a) Try browsing files in both the terminal and the graphical file browser:

- **Open** a new terminal using either of the methods mentioned above. (You should already have a graphical view of your home folder open from the previous exercise.)
- In the terminal, type the command **ls**. Compare what you see listed with what you see in the graphical representation of your ***home*** directory.
- Type the command **ls –l** and note the kind of information being provided and how it compares to the graphical representation of your files.
- In the graphical File Browser, click on the List option under the View menu, and compare this information to that provided using the **ls –l** command.
- In the console, type **ls –l bioinf_files .** Click on the **bioinf_files** folder in the graphical file browser and compare what you are seeing.

You can also use **glob patterns** to limit the files you wish to list.

*****	an asterisk means any string of characters
?	a question mark means a single character
[ ]	square brackets can be used to designate a group of characters

More details about this are given in the ***Linux/Unix shorthand and shortcuts*** **section below.**

(Exercise 1-3, continued)

b) Try these commands that use wildcards to match multiple files:

- List all the files in the directory **bioinf_files**. that start with the letters **tes**

```
ls bioinf_files/tes*
```

- List all the files in your directory that start with tes, and end in 1.embl, 2.embl or 3.embl

```
ls bioinf_files/tes*[1-3].embl
```

Learning about Linux commands

Most Linux commands have a manual page that provides information about the command and options that can alter its behaviour. Many tasks can be made easier by using command options. A good rule of thumb is to ask yourself whether what you want to do is something many others may have wanted to do. If the answer is yes, then there may well be commands and options available to do that task.

Linux manual pages are referred to as **man pages**. To open the man page for a particular command, you just need to type **man** followed by the name of the command you are interested in. To browse through a man page, use the cursor keys (↓ and ↑). To close the man page simply hit the **q** key on your keyboard.

If you do not know the name of a command to use for a particular job, you can search using **man –k** followed by the type of thing you are trying to do. An example of this is in exercise 1-3, part c).

(Exercise 1-3, continued)

c)

- Look up the manual information for the **ls** command by typing the following in a terminal:

```
man  ls
```

- Read through the man page. You can scroll forward using the up and down arrow keys on your keyboard. You can go forward a page by using the space bar, and move backwards a page by using the **b** key.

- What does the **-h** option do? What about the **-a** option? What would running **ls -lrt** do?

- Press the **q** key when you want to quit reading the **man** page.

- Try running ls using some of the options mentioned above.

- Look up some programs with man pages with the keywords “list directory”

```
man –k “list directory”
```

You could now look at the **man** *pages for any Linux commands used in this tutorial to learn about what they do and any options that could be useful to your work.*

Basic Linux/Unix tips for filenames

a) Certain characters should not be used in filenames in Linux/Unix.

If you stick with letters, numbers, hyphens, underscores and full stops, you will be fine.

b) Linux/Unix does not deal well with spaces in filenames!

Make sure your filenames do not contain them.

Filenames with spaces in them are a common problem when transferring files to Linux/Unix from computers running Windows, or Mac operating systems.

If you end up with filenames with spaces in them, you will need to enclose the entire filename in quotation marks so that Linux/Unix understands that the space is part of the name.

Alternatively, you can "escape" the space using a backslash. For example, if I have a file called

```
my document
```

Linux/Unix will see this as two filenames, "my" and "document".

But you could write either of the following to make it understand you mean a single file:

```
“my document”
my\ document
```

Our general advice is to change the name of such files to remove the space. A common practice is to replace the space with an underscore. For example:

```
mv “my document”  my_document
```

c) Assume that everything is case specific
Linux/Unix systems consider capital letters different from lower case letters. The filename **myFile** is not the same as the filename **Myfile** or **myfile**.

Please note that there are some common naming conventions in place for biological data that you should try to follow. More is said on this in the second part of this tutorial.

Getting the prompt back when running graphical applications from the terminal

On an earlier page the command **gnome-terminal &** was suggested as a way to start a new terminal, but the ampersand symbol was not explained. Ending a command with **&** tells the shell to go immediately back to the prompt, not waiting for the command to complete. It is generally used where you expect the command to open up a new window or where you expect the command to run for a long time in the background and you want to keep working while it runs. It is also possible to change your mind and get the prompt back while the command is running.

Confusingly, some graphical programs will always signal the shell to keep going even if you omit the **&** from the command. To demonstrate the default behavior we can use a very simple toy command called **xeyes.** The following exercise will help you understand how all this works.

Exercise:

1. In a terminal, type the command **xeyes**
    1. A small window should open with some eyes that follow the mouse. Move the mouse around it.
    2. Try to type another command (eg. **pwd**) back in the same terminal.
    3. Close the Xeyes window and now see what happens back in the terminal.
2. Run **xeyes** again and leave it running
    1. Back at the terminal, type **Ctrl-z** (ie. hold down Ctrl and tap z).
    2. Now try typing another command.
    3. Try moving the mouse over the eyes.
    4. In the terminal, give the command **bg** and try moving the eyes again.
3. Run xeyes once again with an ampersand after the command – **xeyes &**

Linux shorthand and shortcuts

Understanding Linux commands can seem daunting at first. This is in part due to particular characters (full stops, question marks, etc.) having special meaning in commands. Once you learn the basics, these shorthand characters are extremely useful and time saving.

The following incomplete list covers the symbols you will see most often today and describes their meanings as you will most likely encounter them in this course.

***** matches any character appearing 0 or more times, also known as a wildcard

ls mydir/* *list all the files under the directory mydir*
ls cat* *list all files starting with the letters* cat
ls cat*hat *list all files starting with the letters* cat *and ending in* hat

? matches a single character

ls cat??hat *list all files starting with the letters* cat *followed by any 2 letters, and then* hat

. the directory you are currently in – ie. the last one you moved to using **cd**

.. the directory one level above the one you are currently in, aka. the parent directory

~ shorthand for your home directory, eg. /home/live

$var dollar sign indicates a variable substitution, even within double quotes
– see the section on environment variables

! used for history substitution – see p21

- apart from preceding a parameter (eg. **ls -l**) it can also signify stdin/stdout;
also, the command **cd -** is a special case meaning “cd to previous directory”

; a semicolon can be used to separate two commands on the same line;
it is also used when writing loops – see p59

A warning: some symbols take on very different meanings depending where they are used.

More Basic Linux Commands

A list of common Linux commands is provided in ***Appendix D*** *of this document for reference.*

Changing directories

The command used to change directories is **cd**

If you think of your directory structure, (i.e. this set of nested file folders you are in), as a tree structure, then the simplest directory change you can do is move into a directory directly above or below the one you are in.

To change to a directory one below you are in, just use the **cd** command followed by the subdirectory name:

```
cd  subdir_name
```

To change directory to the one above your are in, use the shorthand for "the directory above"   **..**

```
cd ..
```

If you need to change directory to one far away on the system, you could explicitly state the full path:

```
cd /usr/local/bin
```

If you wish to return to your home directory at any time, just type **cd** by itself.

```
cd
```

And finally, you can type

```
cd –
```

This returns you to the last directory you were working in before this one.

If you get lost and want to confirm where you are in the directory structure , use the **pwd** command (*print working directory*). This will return the full path of the directory you are currently in.

Note also that by default in Bio-Linux, you see the name of the current directory you are working in as part of your prompt.

For example, when you first opened the terminal in a live session you should see the prompt:

```
live@biolinux[live]
```

This means you are logged in as the user **live** on the machine named **biolinux**, and you are in a directory called **live**. (Recall that your home directory is /home/live.)

If you move into the **bioinf_files** directory

```
cd bioinf_files
```

you would see the prompt:

```
live@biolinux[bioinf_files]
```

Exercise 1-4

- Change directory from your home directory to the directory bioinf_files by typing

  cd bioinf_files

- Find the full path to where you are by typing

  pwd

- Go back to your home directory by typing

  cd

- Change directory into the /usr/local/bin directory by typing

  cd /usr/local/bin

- List the files in this directory.

  Many of the programs in **/usr/local/bin** *are* ***bioinformatics*** *programs.*
  The main directory of programs on the system is /usr/bin.
  Some bioinformatics software can be found in there also.

- Go back to your home directory by typing

  cd

Tab completion

Tab completion is an incredibly useful facility for working on the command line.

One thing tab completion does is complete the filename or program name you want, saving huge amounts of typing time.

For example, from your home directory, you could type:

cd bio

and hit the tab key.

If there is only one directory with a name starting with the letters “bio”, the rest of the name will be completed for you. Here this would give you:

cd bioinf_files

User accounts on Bio-Linux are setup such that if there is more than one file with that combination of letters, all the files will be shown to you. You can choose the one you want by typing more of the filename, or by continuing to hit the tab key.

Exercise 1-5

- Return to your home directory if you are not already there by typing **cd**
- Type **cd bio** and use tab completion for the rest of the command. Then press the **return** key.
- You will now be in the **bioinf_files** directory.
- Type **ls testseq** and use **tab** completion. This will show you a list of files that start with *testseq*.

You now have the option of completing the filename yourself, or "tabbing" through the filenames available.

- Press the **tab** key a number of times to see what happens.
- Type **ls c** and use tab completion to view the files available.
- Type an **a** such that you now have **ls ca** on the command line.
- Now press the **tab** key again. You can gradually add extra letters and use the **tab** key to limit the options available.

As you get faster with this, it will save you a lot of typing effort.

Exercise 1-6

- Type **a** on the command line and then press the tab key.
- Add **rte** to the **a** so that you now have **arte** on the command line. Press the **tab** key again.
- You will see that there is only one command that starts with these letters: **artemis**

For programs that might contain case sensitive names, tab completion can be especially useful.

- Type **bl** on the command line and press the **tab** key. You will see a number of program names listed.
- Keep pressing the tab key to see how the filenames will cycle through on the command line.

Note on tab completion of commands

The default tab completion capabilities on Bio-Linux allow you to list the program names available on the system by typing the first letter or letters of the name at the prompt. This works because the system knows that the first item typed on the command line is a command (recall the Anatomy of a Command section above).

Command history

Previous commands you have used are stored in your history. You can save a lot of typing by using your command history effectively. If you use the up arrow key when you are at the prompt in your terminal, you can see previous commands you have run. This is particularly useful if you have mistyped something and want to edit the command without writing the whole command out again.

You can also view past commands using the command **history**. By default, **history** will return a list of the last 15 commands run. You can add a number as a parameter to the command to ask for longer or shorter lists. For example, to return the last 30 commands run, you would type:

```
history -30
```

To re-run a command listed by the **history** command, you can just type the command number, preceded by an exclamation mark. E.g. to run command number 12 returned to you, you can type:

```
!12
```

It is also possible to "speed search" previously-executed commands by typing the first few letters of the command followed by the key combination:

Ctrl-r (ie. hold down Ctrl and tap the R key)

The command history will be scanned and the matching commands will be displayed on the console. Type **Ctrl-r** repeatedly to cycle through the entire list of matching commands.

Exercise 1-7

- Type **history** on the command line.
- Run one of your previous commands using **!** followed by the number of the command.
- Type **Ctrl-r**, then type **h**.

Making a directory

To make a new directory, use the command **mkdir** (make directory). For example:

```
mkdir newdir
```

would create a new directory called newdir.

Exercise 1-8

- Start in your **bioinf_files** directory.
- Make a new directory called **testdir** (Answer 1a in Appendix A)

  The graphical view of your account should immediately update to show this new directory.

- Move into the new directory **testdir** (Answer 1b in Appendix A)
- Using information in the **Linux/Unix shorthand and shortcuts** section, try to move back into the **bioinf_files** directory using a single command. (Answer 2 in Appendix A)

Office software

There are a number of word processors and spreadsheet programs available on your system. In this course we will look primarily at the LibreOffice suite of programs, previously known as OpenOffice. This is an open source alternative to Microsoft Office and can be run on both Linux and Windows.

The programs within LibreOffice can be run graphically from the icons in the Dash toolbar.

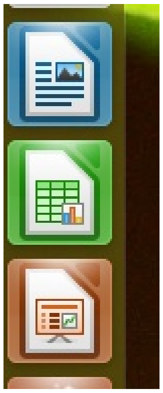


Figure 12: LibreOffice Applications in the dash toolbar

Notes on LibreOffice

LibreOffice programs are very much like MS Office programs in their style and the options they present.

LibreOffice programs can open newer Microsoft formatted files such as Word files (.docx), Excel files (.xlsx) and PowerPoint files (.pptx) and save into those formats also.

OpenOffice will save in its own native formats by default if you create a new file, but you can also save in Microsoft Office formats or export as pdf, among other options.

When saving a Microsoft format file, or choosing to save an OpenOffice file in a Microsoft format, you will be presented with a message warning you that you may lose formatting or content. Your files will generally be fine if they are documents with basic formatting like headings and simple tables, or spreadsheets without macros. LibreOffice is generally happier working with the older .doc, .xls and .ppt formats, so use these in preference to .docs etc.

By default, files will be saved in the same directory you opened the file from, not your Home folder.

Exercise 1-9

- Click on the LibreOffice Calc Spreadsheet icon.
- Under the **File** menu, click on **Open**.
- Look inside the **bioinf_files** directory.
- Open the file called **example.xls**.
- Make a few changes and save the file using the **Save** or **Save As…** options under the **File** menu.
- Close LibreOffice Calc by choosing **Exit** from under the **File** menu.

Using text editors

Plain text files are important, both as input to bioinformatics programs and as input or configuration files for system programs. We highly recommend that you learn to use a **text editor** to prepare and edit plain text files.

Text files, Word Processors and Bioinformatics

Documents written using a word processor such as Microsoft Word or LibreOffice Write are not plain text documents. If your filename has an extension such as .doc or .odt, it is unlikely to be a plain text document. (Try opening a Word document in notepad or another text editor on Windows if you want proof of this.)

Word processors are very useful for preparing documents, but we recommend you do not use them for working with bioinformatics-related files.

We recommend that you prepare text files for bioinformatics analyses using Linux-based text editors and not Windows- or Mac-based text editors. This is because Windows- or Mac-based text editors may insert hidden characters (called carriage returns) that are not handled properly by many Linux-based programs and have to be stripped out.

There are a number of different text editors available on Bio-Linux. These range in ease of use, and each has its pros and cons. In this practical we will briefly look at two editors, **nano** and **gedit**.

Nano

Pros:

- very easy – For example, command options are visible at the bottom of the window
- can be used when logged in without graphical support
- fast to start up and use

Cons:

- by default it puts return characters into lines too long for the screen (i.e. using nano for system administration can be dangerous!) This behaviour can be changed by setting different defaults for the program or running it with the **–w** option.
- it is not completely intuitive for people who are used to graphical word processors

Gedit

Pros:

- very easy
- quite intuitive
- colouring schemes are available for programming syntax
- Very similar to a word processor, but is in fact a powerful text editor.
- **Gedit** has many plugins that are very easy to install and very useful for editing text files and for programming

Cons:

- it is a graphical program and cannot be run from a text-only environment
- it is slightly slower to start up than non-graphical editors

As most users will work on Bio-Linux using a graphical environment, we will only use **Gedit** in the exercise for this section.

Exercise 1-10

Editing a file with Gedit

To start up Gedit, you can use the command line, or find it in the Dash menu. ***Choose one of the two methods*** to open gedit:

Command line

Type **gedit &**

Graphical menu

Click the **Dash Home** at the top left of the screen, then type **edit** and click the ***Text Editor*** icon.

- Type three or four lines of text into the **gedit** window.

- Save your file using the save option under the ***File*** menu (*note, you have to move your mouse right to the top of the screen to see this*) or simply click the ***Save button*** on the ***Toolbar***. Save it as **myfirstfile.txt** in your **testdir** directory.

Exercise 1-10 continued

To save a file under the **testdir** directory, you may have to click on the drop down arrow to Browse for other folders. This will expand this section into a File Browser like the one you've seen in past exercises. Simply browse through to the location **testdir** is in and click the ***Save button***.

- Add a new line to your file and save the file again using the ***Save As…*** option under the ***File*** menu. Save this file as **mysecondfile.txt** in the **testdir** directory.

- Add more functionality to **gedit** by choosing the menu options; ***Edit → Preferences***. A pop-up box will appear with 4 tabs:

View **Editor** **Font & Colours** **Plugins**

Seeing the line numbers in a file helps to keep track of your position in that file. We will enable line numbers here.

- On the View tab enable ***Display line numbers***. Now you can see the line numbers on the left.

- Next, click on the Plugins tab and enable the ***Change Case*** and the ***Document Statistics plugins***. Browse around the other plugins and see what functionality they provide.

- Under the ***Tools*** menu, click on ***Document Statistics***.

- Try out the other newly added plugin, by selecting a piece of text from the document you are editing with the mouse and click on the ***Edit*** menu. Hover the mouse over the **Change Case** menu and choose one of the options you are presented with.

- Change part of one of the lines in this file and save it again using the ***Save As…*** option under the ***File*** menu. This time save it as **mythirdfile.txt** in the **testdir** directory.

- Quit **gedit** by choosing the option ***Quit*** under the ***File*** menu.

Reading text files

There are many commands available for reading text files on Linux/Unix. These are useful when you want to look at the contents of a file, but do not edit them. Among the most common of these commands are **cat**, **more**, and **less**.

cat can be used for concatenating files and reading files into other programs; it is a very useful facility. However, **cat** streams the entire contents of a file to your terminal and is thus not that useful for reading long files as the text streams past too quickly to read.

more and **less** are commands that show the contents of a file one page at a time. **less** has more functionality than **more**. With both **more** and **less**, you can use the space bar to scroll down the page, and typing the letter **q** causes the program to quit – returning you to your command line prompt.

Once you are reading a document with **more** or **less**, typing a forward slash / will start a prompt at the bottom of the page, and you can then type in text that is searched for *below* the point in the document you were at. Typing in a **?** also searches for a text string you enter, but it searches in the document *above* the point you were at. Hitting the **n** key during a search looks for the *next* instance of that text in the file.

With **less** (but not **more**), you can use the arrow keys to scroll up and down the page, and the **b** key to move back up the document if you wish to.

Exercise 1-11

- Move into the **bioinf_files** directory.
- Read the file hsy14768.embl using the commands **cat**, **more** and **less**.

Don't forget that tab completion can save you typing effort.

cat hsy14768.embl

more hsy14768.embl Use the spacebar to scroll down
Press **q** to quit.

less hsy14768.embl Use the ***spacebar*** to scroll down, **b** to go up a page, and the up and down arrow keys to move up and down the file line by line.
Press the / key and search for the letters **sequen** in the file.
Press the **?** key and search for the letters **gene** in the file.
Press the **n** key to search for other instances of **gene** in the file.

For reading files yourself, we recommend the command **less**. The command **cat** is more usually used in conjunction with other commands when you wish to process text from within a file in some way.

Remember the man pages

There are many command line options available for each of the above commands, as well as functionality we do not cover here. To read more about them, consult the manual pages:

man cat
man more
man less

Copying files

The basic command used to copy files using the command line is **cp**. At a minimum, you must also specify the name of the file(s) to be copied, where you wish to copy the file(s) to.

The main things to know about using the **cp** command are:

•if you provide a directory name as the last argument to the command, the files will be copied into that directory
•if you provide a name that cannot be found from your current working directory as the last argument to the command, it will be assumed that this is the filename you wish to use for the file copy.
•if you provide multiple filenames to **cp**, the final filename provided needs to be the name of a directory that already exists – all the files will be copied into this directory.

Examples:

cp firstfile destinationdir	*copy firstfile to a directory called destinationdir*
cp file1 file2 file3 location	*copy file1, file2 and file3 to a directory called location*
cp destdir/* location	c*opy all files in the directory called destdir to the directory called location*

To move whole directories, with all the subfiles and subdirectories, use the **–R** option, (meaning recursive).

cp –R somedir/mydir location	*copy mydir and its contents into the directory called location*

The Linux/Unix shorthand for “this directory right here” (a dot **.**) comes in very handy when copying:

cp –R somedir/mydir .	c*opy mydir and its contents to this directory here*

Make sure you leave a space between the directory name and the shorthand dot.

Also useful is the shorthand for someone’s home account. e.g. instead of having to know and type the location of their account, you can use **~username** In the case of your own account, you use just the ~ symbol, followed by a / if you want to specify any subdirectories in your account.

cp ~user2/somefile .	c*opy the file somefile from user2’s home directory to my current working directory. Note that you need the appropriate permissions to do this!*
cp ~/somedir/mytext .	*copy the file or directory called mytext from within the somedir directory under my home directory, to my current working directory.*

Exercise 1-12

- Move into your directory **testdir**.
- List the files in this directory.
- Make a copy of **myfirstfile.txt** called **test.txt** (Answer 3 in Appendix A)
- Make a copy of **mythirdfile.txt** called **myfourthfile.txt**.
- Make a directory called **subdir**.
- Copy **mysecondfile.txt** into **subdir** (Answer 4 in Appendix A)
- Copy all the files that have the letters **fil** in them into the **subdir** directory. (Answer 5 in Appendix A)
- Move back into the **bioinf_files** directory (Answer 2 in Appendix A)
- Copy all the files that start with the letters **tes** and end in **.embl** into the directory **testdir**. (Answer 6 in Appendix A).

Linking to files

Sometimes you want to access a file or directory at a different location but you don't actually want to copy it. For example if you have a data file in a system folder or network drive that you want to be able to access quickly from your desktop, but you don't actually want the file to be copied to your desktop folder.

```
ln -s  /usr/local/bioinf/sampledata/nucleotide_seqs/multiple_seqs.fasta  ~/Desktop/multiple.fasta
```

Make a link from your desktop to a data file.

If you now try to open multiple.fasta in any application, you will see the data from the linked file as if you accessed it directly. If you write to the link (assuming you have permission) you will be writing data to the original file.

You can examine links using the long output mode of **ls**.

```
ls -l ~/Desktop/multiple.fasta
```

```
lrwxrwxrwx 1 live live 35 2011-05-12 11:46
      /home/live/Desktop/multiple.fasta ->
                        /usr/local/bioinf/sampledata/nucleotide_seqs/file1.fasta
```

The initial letter 'l' shows we are dealing with a link. Links do not have their own permission settings so ls shows them all as enabled, but links do have an owner depending on who created them. The target of the link is shown last. The target can be any file, directory or even another link. Note that Linux will not stop you from making a link where the target is non-existent or inaccessible, but **ls** will help you to spot these “dangling links” by colouring them in red.

Removing files and directories

There are command line and graphical tools for deleting files. You should choose which to use on the basis of what is convenient at the time, and also on the basis of whether you wish to remove the file completely from the system (command line), or whether you like to keep deleted files somewhere for a while, just in case you discover you deleted the wrong thing (default in the graphical interface).

Option 1: Using the command line (effect: deletes files from the system)
To remove a file or files, use the **rm** command followed by the name of the file(s) you wish to delete.

```
rm  file1
rm  file2  file3  file4
rm   cat*                 remove all files starting with the letters cat
```

To remove an ***empty*** directory, you can use the **rmdir** command:

```
rmdir  thisdir
```

If that directory contains any files, you will not able to delete the directory using **rmdir** until you have deleted all the files within it.

To delete a directory and all the files in it at the same time, use the **rm** command with the option **-r** (for recursive)

```
rm –r  fulldir
```

If you use the above command, you will be prompted to confirm that you wish to delete each file. While sometimes useful, this can be tedious. If you are certain that you want to delete all the files in that directory, as well as the directory itself, then you can combine the *recursive* flag with the *force* (**-f**) flag

```
rm -rf anydir
```

So if you are 100% confident that you will never make a mistake, you can use **rm -rf** for all deletions, but for mere mortals it is good practise to try and use the more specific commands, as this can mitigate mistakes.

Option 2: Using the File Browser (effect: moves files into the Wastebasket)
If you view your files graphically, just find the file you wish to remove, right click on it and choose the *Move to Wastebasket* option. Note that this file will not be removed from your system, only moved into a folder allocated for files you no longer want, and can be retrieved via the Wastebasket icon in the bottom right of the screen.

Option 3: Using the File Browser to completely remove files from the system
If you have a graphical view of your files, you can remove the file from the system permanently by finding the file you wish to remove, highlighting it by clicking on it, pressing the Shift key on your keyboard and, while keeping this key depressed, press the Delete key on your keyboard. A message box will pop up asking you to confirm that you wish to permanently delete your file.

Exercise 1-13

- Move into the **testdir** directory.
- Delete **mythirdfile.txt** using the command line (Answer 7 in Appendix A)
- Delete **myfourthfile.txt** using one of the graphical options (options 2 or 3 above). Did you choose to permanently remove the file, or move it into the Deleted Items Folder?
- Move back into your home directory.
- Delete **myfirstfile.txt** from **testdir** without moving to the **testdir** directory. (Answer 8 in Appendix A)
- Delete the entire **subdir** directory from within **testdir** without being prompted about the deletion of each file individually. (Answer 9 in Appendix A)

Notes on Reading, Copying and Removing Files and Directories

On Bio-Linux the commands **cp**, **mv** and **rm** have been aliased to **cp –i** , **mv –i** and **rm –i** respectively.

This means the system will ask you if you really mean to overwrite files should the situation arise with **cp** or **mv**, or delete the file you have just asked to delete when using **rm**. You must respond with a **y** or **Y** if you do wish to proceed. Hitting any other key will cause the action you requested to be ignored.

You cannot assume that other Linux/Unix systems you work on will be configured this way.

Piping and outputting to files

An incredibly powerful facility on the Linux/Unix command line is the ability to take the output of one command and use it directly as the input to another command. This is referred to as "piping" the output of one command into another command.

The vertical bar symbol used for this is called a pipe and looks like: **|**

Standard UK PC keyboards have the pipe symbol on the same key as the backslash symbol, at the bottom, left hand side of the keyboard. Pressing the Shift key and the backslash key together will give you the pipe symbol.

On some keyboards, the pipe symbol is at the top left hand side, on the same key as the backtick. To type a pipe symbol on such keyboards, hold down the key **Alt Gr** and hit the back tick (`) key (left of the number 1 key).

An example of when you might use a pipe would be if you wanted to list all the files in a directory, but there are too many to fit on a single page. This is probably what you saw when you listed the contents of /usr/local/bin in Exercise 1-4.

You can **pipe** the output of the **ls** command (a list of files) into the **less** command, which will allow you to view the list page by page. To list the files in /usr/local/bin and view them page by page, the command would be written:

```
ls  /usr/local/bin  |   less
```

Another useful command is the **wc** command, which stands for wordcount. By default, **wc** returns the number of newlines, words and bytes in a file (or in information given to it via a pipe). Using command line options, you can get **wc** to return just the number of lines, just the number of words or just the number of bytes.

There are other options available for obtaining information from a file that can be found by reading the manual page for ***wc.***

For example, you could find out how many files you had in a directory by typing:

```
ls  |  wc -l
```

Diff, Grep and Sort

In this section, we look briefly at three very useful commands: **diff**, **grep** and **sort**. As with all the commands covered today, we recommend that you read the manual page for more information about how these work and what options are available.

Diff

diff compares files line by line and reports the differences between the files. In fact, **diff** can be used for more involved tasks as well, like comparing the contents of directories. This can be very useful when you are looking for changes that you or someone else has made.

Exercise 1-14

- Move into the **testdir** directory.
- Type **diff test.txt mysecondfile.txt** to see what **diff** reports to you.
- Type **cat mysecondfile.txt | diff - test.txt**

In the above command the hyphen (**-**) refers to the information being given to **diff** from the pipe. That is, the information resulting from the command **cat mysecondfile.txt** is put directly into the diff command. Obviously, in this instance it would be easier just to give the name of the file, **mysecondfile.txt**, but there are many instances where being able to use – to mean “what I am sending via a pipe” can be extremely useful.

Grep

grep stands for **global regular expression print;** you use this command to search for text patterns in a file (or any stream of text). Eg.

```
grep  “adge”  /usr/share/dict/words
```

You can also use flexible search terms, known as **regular expressions**, in your grep searches. You have already used glob pattern expressions in this practical, but regular expressions are somewhat different and more powerful. For example, when you listed all files with the pattern **tes*embl*** you were using a glob pattern comprising explicit characters (e.g. **tes**) and special symbols (* meaning any character or characters). The equivalent in **grep** would be **“tes.*embl.*”** where the period signifies any single character and the * signifies any number of repeats.

Therefore to convert from a shell glob pattern to a regular expression replace each * with **.*** and each **?** with **.** . You also need to enclose the expression in quotes to tell the shell not to try and interpret it as a glob.

Unmodified glob patterns will be accepted by grep but will not work as intended. For example the pattern **tes*** in **grep** means **te** followed by any number of **s** characters in sequence **(te, tes, tess, tesss, ...)**. The question mark now signifies optionality – so **tes?** means **te** followed by zero or one **s** character **(te, tes)**. Regular expressions are found in several places other than **grep**, most notably in the Perl scripting language. The full syntax is extensive and powerful but is beyond the scope of this course, so back to the grep command itself...

grep requires a regular expression pattern as a parameter, and prints all the lines in a file containing that pattern.

grep is especially useful in combination with pipes as you can filter the results of other commands.

For example, perhaps you only want to see only the information in an EMBL file relating to the origin of the sequence, that is, the DE line. You do not need to search the file in an editor, you can just **grep** for lines beginning in DE, as in the next exercise.

Exercise 1-15

- While in the **bioinf_files** directory, type the command: **grep "DE" hsy14768.embl**

  What is this command doing?
  Can you see why the above command results in the output you see?
  An explanation of this command can be found below this exercise box.

- Try the commands: **grep "^DE" hsy14768.embl** and **grep -x "DE.*" hsy14768.embl**

  What are the ^ symbol and the -x parameter in these commands doing?
  Check the manpage for ***grep*** *to be sure.*

- Try the command: **cat hsy14768.embl | grep "^DE"**. Does that do what you expected?

- Move to your home directory and type **ls –lR**

  Read the manual page for ***ls*** *if it is not clear what this command returns.*

- Use the above command with a pipe and a **grep** command to search for files created or modified today.

- List the files in the **bioinf_files** directory and use the **grep** command to look for those containing the characters **d4**.

The first command in the above exercise searches all the text in the hsy14768.embl file and returns the lines in which it finds the letter D followed by the letter E.

The second command in the exercise also returns lines in the file that have a letter D followed by a letter E, but only where DE is found at the beginning of a line. This is because the ^ symbol means "match at the beginning of a line". The **$** symbol can be used similarly to mean "at the end of a line". These are known as **anchors.** Passing the **-x** flag to **grep** tells it to automatically anchor both ends of the search pattern.

What this anchoring does in the example above is return to you just the organism information in the embl file. This is because none of the other lines returned in the previous command started with DE, they just contained DE somewhere in them. This is an example where knowing how information is stored in an given file, along with a few basic Linux commands, allows you to retrieve information quickly.

Another common example is counting how many sequences are in a set of multi-fasta files. We can do this with **pipes** between the commands **cat**, **grep** and the handy **wc** utility, which here we use to count lines found by grep.

```
cat *seqs.fasta  |   grep "^>"  |  wc  -l
```

Each sequence in a fasta file starts with a header line that begins with a > . The above command streams the contents of all files matching the glob pattern *seqs.fasta through a search with **grep** looking for lines that start with the symbol > . The quotes around the pattern ^> are necessary, as otherwise it is interpreted as a request for redirection of output to a file, rather than as a character to look for. As before, the ^ symbol means "match only at the beginning of the line".

The output of this **grep** search is sent to the **wc** command, with the **-l** indicating that you want to know the number of lines – ie. the number of headers and by implication the number of sequences.

So a synopsis of the command above is: *Read through all files with names ending seqs.fasta and look for all the header lines in the combined output, then count up those lines that matched and return the number to screen.*

We cover sequence formats later on in part 2 of the tutorial.

Environment Variables

We have seen that the way commands run can be modified by the options passed on the command line. Some commands also read values called environment variables which affect their behaviour. Environmental variables are set within the shell via the **export** command and are passed to any processes you run. This is useful when you want to set some parameter that is common to all invocations of a command, or applies across several commands. For example, your favourite text editor may be, say, Gedit, or Nano, or Vim, or Emacs. In the shell you can say:

export EDITOR=vim

Now any command that wants to run a text editor knows what your preferred editor is. Within the shell you can get at the current value of en environment variable by prefixing it with a **$** sign, eg.

echo $EDITOR *prints the current value of the EDITOR environment variable to the screen*

The **printenv** command dumps all environment variables. Note that environment variables are only set in the current shell and are not saved by default, so if you run a command in another terminal or close and restart the terminal any values you set will be lost. For information on making the settings permanent by editing your **.zshrc** file see the user guide under *Supported Shells*.

Exercise 1-16

- Give the command: **export VAR1=hello**, then:
  - **echo $VAR1**
  - **echo $ VAR1**
  - **echo “$VAR1”**
  - **echo '$VAR1'**

- Start a new shell by typing: **gnome-terminal &**
  - Within this new shell: **echo $VAR1**

- Start a second new shell by selecting Terminal from the Applications->Accessories menu
  - Within this new shell: **echo $VAR1**

- Go back to the original shell window
  - **unset VAR1**
  - **echo $VAR1**

- Has this affected either of the other two shells you started? Check them:
  - **echo $VAR1**

Changing permissions on files and directories

Every file on the system has a set of permissions on it that dictate who on the system can read, change or delete, or execute the file. By default, all the files you create in your account are readable, changeable or executable by you. However, you can grant other users permissions to access parts of your account if you wish.

Below is some basic information about file permissions. To set up access to your files for other people on the system, please get advice from your system administrator.

The command to change permissions is **chmod**. You have to specify who you are modifying the permissions of, what the new permissions are, and what file or directory to act on.

The format of the chmod command is:

chmod who ± permissions filename

who can be:

u means **user** and refers to the owner of the file
g means **group**, and refers to the group the file belongs to
o means **others**, everyone on your systems apart from those above
a means **all** three, i.e. user, group and others

permissions can be:

r means **read** permission
w means **write** permission
x means **execute** permission

Each user has a default group and possibly extra group memberships. Use the **id** command to view your group memberships. When you create a new file it will be owned by you and by your default group. If you are a member of additional groups, you can switch the file to any of those groups using the **chgrp** command. (Please refer to the manual pages for the commands **chown, chgrp** and **chmod** for more on this topic.)

For simplicity, let us assume that you and a co-worker have both been put in the default group **labusers** and wish to share your data files found in ~/bioinf_files.

chmod a+x ~ give permission to anyone to execute, in this case, so that they can move through, your home directory.

chmod g+rx ~/bioinf_files give permission to people in the group to access files in the bioinf_files directory under your home directory, including l isting the files with **ls**

chmod g+r ~/bioinf_files/* give permission to people in the group to read the files in the directory

The first command could have been "**chmod g+x ~".** This would unlock your home directory only to users in the **labusers** group. However, enabling access for anyone is generally safe, as permissions on all the files and subfolders prevent anyone from actually accessing them, and unless you set **a+w** in addition to **a+x** nobody but you will be able to list the files in your home directory.

Some other useful information

Copying and pasting text

Most Linux applications, including the shell terminal windows, have Copy and Paste options in the Edit menu or available in the pop-up menu when you click the right mouse button. You can copy text within the application or between different applications. There is also a quick way to copy text within the terminal by ***highlighting text to select it, and using the middle mouse button to paste the text.***

The exact way to select, copy and paste text from within a terminal windows depends on how your mouse has been set up. Normally you would highlight text by dragging the mouse across it with your left mouse button depressed to copy the text, and paste by clicking the middle mouse button (or the two outer mouse buttons pressed simultaneously). Note that within the terminal it doesn't matter where you click the middle mouse button – the text will always be inserted at the current cursor position.

The simple way to stop a process

Sometimes a command or program you run in the terminal goes on too long, or is obviously doing something you did not plan. If there is no obvious way (such as a menu option or button) to stop the program running, try using **Control-c** (more commonly written as **Ctrl-c**). i.e. hold down the **Control** key and hit the **c** key. This requests the program to stop immediately, though the program may ignore the request.

Note that this is the same key combination used in most graphical applications for copying text. Remember that highlighting text in a Linux terminal automatically copies it into the buffer – you don't need to press Ctrl-c before pasting with the middle button.

Putting a command to one side

Sometimes, you are in the middle of typing a long command, and you suddenly realise you need to do something else in the terminal, like list the current directory contents or check the manpage, before you run the command. Z-shell provides a handy shortcut for this: **Alt-q**. When you press **Alt-q** the current command disappears and you have a new empty prompt, but the unfinished command has been remembered and will reappear with the next prompt ready for you to edit and run it.

An alternative is to hit **Ctrl-c**. Within the shell, **Ctrl-c** does not cause the shell to exit but it does cause the current command to be abandoned and a fresh prompt to appear. Unlike with **Alt-q** the unfinished command will still be visible in the terminal display so you can select it and paste it back in with the middle button if you decide you want it after all. (Try it!)

Logging out of a session

To logout, you can press the ***Power Icon*** on the far right of the top taskbar (Figure 2) and choose the ***Log Out*** option.

To shut down the machine, you can choose the ***Shut Down*** option on the same menu. If you are working on the console of a machine with users apart from you, then please check with your system administrator before powering down the machine. Other people might want to log in remotely. In addition, Bio-Linux machines are configured by default to update system and bioinformatics software overnight. If your machine is turned off every night, it cannot update this way.

Clearing your terminal of text

Your terminal windows can fill up with lots of text, and it can become difficult to see the information you want because of all the clutter. You can clear the terminal window of all previous text by typing

```
clear
```

Accessing a running program or working with others interactively

If you just run a job and then close down the terminal you ran it from, often the job will be terminated. It would be nice to be able to leave a long job running and be able to log out and then log back in again to see how it is progressing. This is especially true if you work remotely and experience network disruptions, or if you run programs that can take quite a long time, but ask you for input periodically.

Luckily, there is a tool that makes it possible to leave programs running with no danger of them terminating if you log off or your terminal is closed. In addition, when you log back into your system, either locally or remotely, you can "re-attach" to your earlier session so it feels like you are picking up where you left off, in the same window you were running your program from.

The utility that allows you to do this is called **screen**. It must be run before you start running other programs in your window. **Screen** can also allow two people on different machines to work in the same session – i.e. Real time collaborative editing is possible with **screen**.

Unfortunately, how to work with screen is beyond the scope of this course. However, the link below provides links to a number of tutorials about screen and multi-user sessions:

http://www.xmarks.com/topic/gnu_screen

An extensive list of command options can be found at:

http://www.oreillynet.com/linux/cmd/cmd.csp?path=s/screen

Accessing a full graphical desktop remotely

You can connect to your (installed) Bio-Linux system remotely using NX software. If you download an NX client to another Windows or Linux system, you can connect to an installed Bio-Linux system and run a full, graphical, desktop session remotely. Further details on how to do this can be found on the NEBC website at:

http://nebc.nerc.ac.uk/tools/bio-linux/accessing-bio-linux

Note that due to limitations of the remote protocol, NX will use a fallback desktop "Gnome" session which is slightly different to the default "Unity" desktop environment described in this tutorial.

There are many useful commands available on ***Linux*** and we cannot begin to cover them in this course. We recommend that you consider buying a book to help you learn how to use ***Linux*** efficiently.

Part Two: Introduction to Bioinformatics on Bio-Linux

This section of the tutorial introduces you to running bioinformatics software on Bio-Linux, including how to find out what is available for particular types of bioinformatics tasks, some options you have for running programs on the system, and where to find documentation about the software on the system. This course does not cover the detailed use or understanding of any particular piece of software.

The main points we hope you take away after completing this section of the tutorial are:

a) If you have repetitive tasks to carry out, chances are there are ways of fully or partially automating them.

b) Web interfaces are easy, and have certain benefits, but there are other ways to access software, and sometimes these will suit your needs better.

c) If you are funded by the NERC, we can be contacted directly for help. **Please email us if you have questions or problems relating to Bio-Linux or bioinformatics analysis.** Our contact address is helpdesk@nebc.nerc.ac.uk

Documentation and Help for Bioinformatics Software on Bio-Linux

There are a number of sources of information about the bioinformatics software on Bio-Linux, including

- Bio-Linux bioinformatics documentation
- local copies of software documentation
- options under the help menus in some graphical programs
- web pages
- journal articles.

Bio-Linux Bioinformatics Documentation

Categorised information about all bioinformatics software on the Bio-Linux system can be accessed via the **Bioinformatics Docs** icon on the left hand side of your desktop. Software can be listed by name or by functional category.

The information for each program includes an overview of what it does, links to local documentation when available, as well as links to information on the internet.

We highly recommend that you read the documentation for any programs you intend to run.

This is especially important for programs that use heuristic algorithms (methods involving some level of approximation, such as BLAST), and those that output numerical results.

Exercise 2-1

- Click on the ***Bio-Linux Documentation*** icon on the desktop, then on ***Bioinformatics Docs***

- Select a category under the ***Browse by Category*** section.

- Click on the names of any of the programs that might interest you and view the information in the resulting web page.

- Return to the search form and click on the link to ***List all categories***. This shows a view of all the documented software according to the functional category (or categories) they are listed in.

Please refer to the bioinformatics documentation throughout this tutorial to find out more about the programs introduced.

If you know of a good information resource for a program on Bio-Linux that is not mentioned in our bioinformatics documentation system, or you have any problems with the system, please let us know by emailing us at helpdesk@nebc.nerc.ac.uk.

Help Functions within the Programs

Documentation is available from within many programs. For example, many graphical programs have a Help menu or button; many command line programs provide help if you type the name of the program followed by **–h**, **–help** or **--help**. Some programs even have their own manual pages that can be accessed by typing **man** followed by the program name.

Example data for this tutorial

The sequences referred to in this tutorial can be downloaded in a compressed package from

http://nebc.nerc.ac.uk/downloads/courses/Bio-Linux/bioinf_files.tar.gz

If you have *just done* the associated Introduction to Linux tutorial, you will *already have* these files – please move on to the next section of the tutorial.

If you have *joined the tutorial at this point*, please refer to Exercise 1-1, parts b, c and d to download and unpack the necessary sample data files.

Interface choices

Software can be run on the command line, via graphical programs on your computer, via web interfaces, via web services and/or via scripts. Bioinformatics programs can often be run using more than one of these options. Each type of interface has pros and cons. We have summarised some of these for reference below.

Interface	*Pros*	*Cons*
Command line *Type out the command and press enter*	Fast to run once you know the program Very flexible; usually many options Repetitive tasks are easy to run or automate Easy to log in remotely and carry out tasks	Have to learn the syntax Have to find out what options are available
Prompted command line *Type out the command and respond to prompts on screen*	Easy to run; don't have to remember the command line syntax Easy to log in remotely and carry out tasks	Easy to forget the diversity of options for a program because of the temptation to just reply to prompts provided Slower to get running than “pure” command line
Graphical interface *Start the program and interact via menus*	Often more intuitive and visually pleasing than the command line Extensive help is often available via a menu option or button Some programs (not all!) can be run by clicking an icon in the Applications | Bioinformatics menu on your system. Appropriate for visual tasks such as alignment editing, detailed annotation checking, etc.	Can be slower to use than the command line, especially for repetitive tasks For some programs, the command line version provides more functionality. You may need your system admin to set up programs so that you can run graphical programs when logging in remotely
Web interface *Run via a web browser window, usually at a remote site*	Usually intuitive Can provide functionality not available via locally-run programs such as access to important data resources or results presented in useful formats, e.g. including links to related data resources, graphics, etc. Some websites allow a certain degree of “pipelining”, where the outputs of one program can intuitively be supplied as input to another.	Can be slow to use relative to the command line, especially for repetitive tasks You are subject to the rules and restrictions of the site you are working on (e.g. data volume, number of tasks, options available, etc.) You may not want to send private data over the internet (e.g. if you are applying for a patent?) You can be subject to the whims of network connectivity
Web services *Runs tasks over the internet from a program, usually locally installed or run via java webstart.*	Can bring together the ease of a locally run program with the data and computing resources of a remote site Can be used via graphical programs or scripts	You are dependent on network connectivity You are dependent on the consistency of the remote server where the functions you need are running You are dependent on the functionality the remote site offers; this may not be as extensive as the functionality you get locally for some programs.
	Very flexible	You have to write the script or find a script that

Scripts *Using a small program that runs a program or programs for you*	Great for automating tasks Great for carrying out customised tasks Straightforward to learn enough to alter existing scripts to do exactly the task you want.	does the job. This means learning a programming language (or asking someone who knows one to help you)

For repetitive tasks, we highly recommend the use of the command line, workflow software and/or scripting.

General points about working with bioinformatics programs

Sequence formats

A simple thing that often trips people up is ***sequence formats***. There are many different sequence formats; the reasons for this are both historical and functional.

Historically, when people first started writing analysis programs for molecular data, they designed a format that they felt suited their needs. As time went on, numerous formats came into existence. We live with the legacy of this. We must know what format our data is in, and whether the program we want to run can use data in that format.

Functionally, a program may require information that can be included with data held in certain formats, but not others. For example, *embl* format files can, in addition to the sequence data itself, contain descriptive information about a sequence, such as its features. In contrast, *plain* format contains nothing inside the file except the sequence data, while *fasta* format allows a small amount of information about a sequence to be given in a header line. *Clustal* and *msf* formats handle multiple aligned sequences, while *phylip* and *nexus* format files contain aligned sequences as well as information relevant to phylogenetic analysis programs.

To analyse data, it must be presented to the analysis program in a format the progam understands.

This seems obvious, but frequent errors (or worse, misleading results) occur when the data entered into a program is not appropriate.

Converting files to different sequence formats used to be a frequent, and often time consuming, task in bioinformatics. Luckily there are file conversion programs that take care of this easily for many formats. In addition, many program understand more than one format.

Some common bioinformatics sequence formats, along with common filename conventions used for those formats, are listed in the table that follows the next section.

We recommend the following page for more information and examples of common bioinformatics file formats:

http://www.molecularevolution.org/resources/fileformats

File naming conventions in bioinformatics

The **suffix**, (the part of the filename after the final dot), is often used to denote to you, and other people, what the format of the data inside the file is.

For example, the common suffix for clustal formatted alignments is ***aln***. .A bioinformatics file that ends in **.aln** is usually assumed to be a clustal formatted alignment file.

Another multiple sequence alignment format is phylip. A common suffix used on files containing sequences in phylip format is **phy**.

Common suffices used for files containing data in particular formats are listed in the table following this section. We highly recommend that you follow conventions when naming your data files.

Benefits to following the convention for filename endings include:

- You will know your data format just by looking at the name of the file.
- Following standard conventions, (rather than making up your own naming system), makes it easier for other people looking at your files, (e.g. collaborators, or people helping you); they will know the data format just by looking at the name.
- Some graphical programs have filters set so that only files with particular suffices will be listed in the file browser window when you try to load some data. If you use conventional filename endings, this is less likely to cause problems for you.

Certain programs use information in the filename to interpret aspects of the data, (not just the data format). Such programs have strict naming conventions for the whole filename. For example, some sequence assembly programs either require, or are benefited by, defined naming schemes for sequence traces. The filename will inform them about which sequences are read pairs, what direction sequence reads are in, and other information relevant to assembly or visualisation. You will need to read the program documentation to find out what is required in such instances.

> You are not restricted to naming your files in any particular way but we ***highly recommend*** that you follow the convention for the type of file you are generating/saving.
>
> Following file naming conventions from the beginning will save you, and your collaborators, ***a lot*** of time!

Common bioinformatics file formats

Format	*Some common filename endings*	*Comments*
Embl or swissprot	.dat .embl .sprot .swiss	Usually these files, along with genbank files, contain feature information as well as sequence. Embl and Swisprot (or Uniprot) format are the same. Embl files contains nucleotide sequences and Uniprot files contain peptide sequences. Files downloaded from EMBL or Uniprot websites use the suffix .dat. Often these are compressed with gzip, and so end in .dat.gz Files generated by individuals in embl format will tend to end in .embl.
Genbank	.seq .gb .genbank	These files, along with embl and swissprot files, usually contain feature information as well as sequence. Individuals using this format, usually use the .gb or .genbank suffix. The NCBI usually uses .seq for genbank sections.
Fasta	.fasta .fsa	Possibly the most common sequence format. It may contain nucleotide or peptide sequence(s) and usually has little in the way of feature information.
Plain	.pln .staden .sdn	Not commonly used, as the file contents contain nothing but the sequence itself; the only identifier of the sequence is in the filename. Staden programs use the plain format, accounting for the last two of the file suffices given.
Clustal	.aln	Multiple sequence alignment format Originally from the clustalw program, but now recognised by many programs that accept or output multiple sequence alignments.
Phylip	.phy .phylip	Multiple sequence alignment format Used by the Phylip suite of programs and many others, especially those associated with phylogenetic analysis.
Msf	.msf	Multiple sequence alignment format This was the standard output format from some of the suite of programs called GCG. The format is still sometimes used. Other multiple alignment formats are more generally used and thus are often a better option to choose if you have a choice.
Nexus	.nxs .nex	Multiple sequence alignment format Used by a number of phylogenetics programs.
GFF	.gff	A format for describing genes and other features associated with DNA, RNA and Protein sequences. Not generally used as input for analyses.

Naming files and the danger of over-writing previous results

Many programs will suggest a name for your results file. Sometimes this name is generated by taking the beginning of the name of your input file, and adding a new suffix. However, sometimes it is just a generic name like *prettyplot.ps* or *clustalw.aln*. We encourage you to ***change generic names*** as soon as you can.

Apart from the fact that filenames like *prettyplot.ps* give you little idea what is in the file, if you do not change the name, **the next time a file of the same name is generated, you will overwrite previous results.**

A common problem: what is a text file and what is not

Sequence data are usually stored in text or binary files. Text files contain human readable data. Binary files are not human readable. The file formats referred to in the table above are all text formats. Examples of binary formats include ABI sequences and SFF sequence files.

Word documents may look like text, but they aren't. The letters you see on the page of a Word document (or OpenOffice Write, or other word processing programs) are stored in a **binary** format.

Most sequence analysis programs expect **text**. Plain old, nothing fancy, text.

It is an unusual situation to need to use sequence data that has been stored as a Word document (if it is not unusual to you, you are probably doing things the hard way!). To get a text document when using Word, save it as **text only**.

Rule of thumb

If you are using Word or any other word processing program at any stage your work with sequences, then it is very likely that your life could be made a lot easier.

Please seek advice about other ways to handle your data. You will almost certainly save yourself time and frustration. Honest.

Exercise 2-2

A useful Linux command to find out what type of file you are dealing with is **file**. This does not look at the filename but interrogates the file contents directly.

- In your **bioinf_files** directory is the file example.xls. Move into your bioinf_files directory if you are not already there and try running the command

```
file  example.xls
```

- In the bioinf_files directory is a file called testseq1.embl. Try running the command

```
file  testseq1.embl
```

Code Corner

Scripts are small programs written in a scripting language such as Perl or Python. Unlike normal binary applications, the program files can be examined and edited directly using a text editor. However, Linux is able to run these text files as if they were normal commands by automatically invoking the appropriate interpreter named on the first line of the script – for example if the first line of a script says:

```
#!/usr/bin/perl
```

Then the script will be run using the Perl interpreter. Writing scripts is beyond the scope of this course, but it is useful to be able to run scripts that others have written.

Code Corner aims to provide useful scripts suitable for use on Bio-Linux. It can be found on the NEBC website (http://nebc.nerc.ac.uk/tools/code-corner/scripts). If you would like to share your scripts on Code Corner, contact us via helpdesk@nebc.ac.uk.

Examples of running bioinformatics programs on Bio-Linux

For each program covered, we include a list of interfaces available. Those interfaces with an asterisk next to them are covered in the tutorial.

Documentation and links for all the software is available via the Bioinformatics Docs web pages discussed earlier.

Analysing sequences with QIIME

QIIME (pronounced ‘chime’) is a pipeline for performing microbial community analysis that integrates many third party tools which have become standard in the field. QIIME can run on a laptop, a supercomputer, and systems in between such as multicore desktops. QIIME is now included in the standard Bio-Linux distribution.

As an example, we will use data from a study of the response of mouse gut microbial communities to fasting (Crawford et al., 2009). To make this tutorial run quickly on a personal computer, we will use a subset of the data generated from 5 animals kept on the control *ad libitum* fed diet, and 4 animals fasted for 24 hours before sacrifice. At the end of our tutorial, we will be able to compare the community structure of control vs. fasted animals. In particular, we will be able to compare taxonomic profiles for each sample type, differences in diversity metrics within the samples and between the groups, and perform comparative clustering analysis to look for overall differences in the samples.

To process our data, we will perform the following steps, each of which is described in more detail in the Data Analysis Steps:

- Filter the sequence reads for quality and assign multiplexed reads to starting samples by nucleotide barcode.
- Pick Operational Taxonomic Units (OTUs) based on sequence similarity within the reads, and pick a representative sequence from each OTU.
- Assign the OTU to a taxonomic identity using reference databases.
- Align the OTU sequences and create a phylogenetic tree.
- Calculate diversity metrics for each sample and compare the types of communities, using the taxonomic and phylogenetic assignments.

- Generate UPGMA and PCoA plots to visually depict the differences between the samples, and dynamically work with these graphs to generate publication quality figures.

What follows is a streamlined version of the exemplary tutorial provided by QIIME (which can be found at http://qiime.sourceforge.net/tutorials/tutorial.html). Further details and parameters on the below commands and many more can be found at this site.

The material was compiled and adapted by Daniel Pass, School of Biosciences, University of Cardiff, for Bio-Linux courses June 2011. Editorialised for QIIME 1.6 by Tim Booth, NEBC.

QIIME allows analysis of high-throughput community sequencing data
J Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D Bushman, Elizabeth K Costello, Noah Fierer, Antonio Gonzalez Pena, Julia K Goodrich, Jeffrey I Gordon, Gavin A Huttley, Scott T Kelley, Dan Knights, Jeremy E Koenig, Ruth E Ley, Catherine A Lozupone, Daniel McDonald, Brian D Muegge, Meg Pirrung, Jens Reeder, Joel R Sevinsky, Peter J Turnbaugh, William A Walters, Jeremy Widmann, Tanya Yatsunenko, Jesse Zaneveld and Rob Knight; Nature Methods, 2010; doi:10.1038/nmeth.f.303

Note: Commands to type are shown in grey boxes like this. Some commands in QIIME are too long to print on one line, so where you see ➡, you need to continue typing the command on the same line.

Preparation

First, we must copy the tutorial data to your home directory and extract it:

```
cd

tar  -xvzf /usr/local/bioinf/documentation/bio-linux/intro_course/qiime_tutorial_data.tar.gz
```

Entering the directory (cd qiime_tutorial_data) and listing the files (ls) will show what was extracted:

Sequences (.fna)
This is the 454-machine generated FASTA file.
Quality Scores (.qual)
This is the 454-machine generated quality score file, which contains a score for each base in each sequence included in the FASTA file.
Mapping File (Tab-delimited .txt)
The mapping file is generated by the user. This file contains all of the information about the samples necessary to perform the data analysis. At a minimum, the mapping file should contain the name of each sample, the barcode sequence used for each sample, the linker/primer sequence used to amplify the sample, and a Description column.
custom_parameters.txt
Structured file which can be customised to easily tune each analysis.
qiime_tutorial_commands_serial.sh
This is a script which will run all of the commands that we are about to see without user input.
Data
This directory contains the reference files required for alignment of the OTUs.

To begin working with QIIME, you must enter the QIIME shell by typing '**qiime**' in your working directory. This has been successful if the prompt changes to end in '**qiime >**'. The commands below will only be recognised within the special QIIME shell.

Assign Samples to Multiplex Reads

The first task is to assign the multiplex reads to samples based on their nucleotide barcode. Also, this step performs quality filtering based on the characteristics of each sequence, removing any low quality or ambiguous reads. The script for this step is split_libraries.py, but before running it we make a directory for all the output:

```
cd qiime_tutorial_data
pwd                     #This should show we are in qiime_tutorial_data
mkdir out               #This makes a directory for the results to go in
split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -o split_library
```

This invocation will create three files in the new directory **split_library/:**

split_library_log.txt
This file contains the summary of splitting, including the number of reads detected for each sample and a brief summary of any reads that were removed due to quality considerations.

histograms.txt
This tab delimited file shows the number of reads at regular size intervals before and after splitting the library.

seqs.fna
This is a fasta formatted file where each sequence is renamed according to the sample it came from. The header line also contains the name of the read in the input fasta file and information on any barcode errors that were corrected.

Processing sequences into OTUs

There are several steps to go through to produce the annotated OTUs from the input sequences, however the following 5 steps can be called using the '**pick_de_novo_otus'** command found at the end of this section.

1. Pick OTUs
Using the seqs.fna file generated from split_libraries.py, the sequences are clustered into Operational Taxonomic Units (OTUs) based on their sequence similarity. This basic command uses the default parameters: uclust matching, 0.97 sequence similarity, no reverse strand matching.

```
pick_otus.py -i split_library/seqs.fna -o out/uclust_picked_otus
```

2. Pick representative
Since each OTU may be made up of many sequences, we will pick a representative sequence for that OTU for downstream analysis. This representative sequence will be used for taxonomic identification of the OTU and phylogenetic alignment. (options: random, longest, most_abundant, first)

```
mkdir out/rep_set           #This makes a subdirectory to store the representative set
pick_rep_set.py -i out/uclust_picked_otus/seqs_otus.txt  -f  split_library/seqs.fna ...
        -o out/rep_set/seqs_rep_set.fasta  --rep_set_picking_method most_abundant
```

3. Assign taxonomy

You can compare your OTUs against a reference database of your choosing. For our example, we will use the default RDP classification system assignment method which comes ready with QIIME, however BLAST is also an option.

```
assign_taxonomy.py -i out/rep_set/seqs_rep_set.fasta -o out/rdp_assigned_taxonomy
```

4. Make OTU table

Tabulates the number of times an OTU is found in each sample, and adds the taxonomic predictions for each OTU in the last column if a taxonomy file is supplied.

```
make_otu_table.py -i out/uclust_picked_otus/seqs_otus.txt ...
        -t out/rdp_assigned_taxonomy/seqs_rep_set_tax_assignments.txt  -o out/otu_table.biom
```

5. Align sequences

Alignments can either be generated de novo using programs such as MUSCLE, or through assignment to an existing alignment with tools like PyNAST. For small studies such as this tutorial, either method is possible. However, for studies involving many sequences (roughly, more than 1000), the de novo aligners are very slow and assignment with PyNAST is preferred.

```
align_seqs.py  -i out/rep_set/seqs_rep_set.fasta -o out/pynast_aligned_seqs ...
        --alignment_method  pynast  -t data/core_set_aligned.imputed.fasta
```

6. Filter alignment command

Before building the tree, the alignment must be filtered to remove columns comprised only of gaps.

```
filter_alignment.py -i out/pynast_aligned_seqs/seqs_rep_set_aligned.fasta ...
        -o out/pynast_aligned_seqs  --lane_mask_fp data/lanemask_in_1s_and_0s
```

7. Build phylogenetic tree command

Produces a newick formatted tree file (.tre) which can be viewed using most tree visualization tools.
Method options: clearcut, clustalw, raxml, fasttree_v1, fasttree(default), muscle

```
make_phylogeny.py  -i out/pynast_aligned_seqs/seqs_rep_set_aligned_pfiltered.fasta -o out/rep_set.tre
```

The above commands are integral to QIIME and further downstream analysis. Once their function and process is understood, the parameters can be set in the custom_parameters.txt file and run sequentially using the workflow script:

```
pick_de_novo_otus.py -i split_library/seqs.fna -p custom_parameters.txt -o out
# Make sure you change the path in the custom_parameters.txt file before running this command
```

Data to information

QIIME has many different ways to visualize and interrogate the data. Here we will explore just a few.

Note: To open a HTML file type:

```
        firefox filename
```

Heatmap

The QIIME pipeline includes a very useful utility to generate images of the OTU table. You can open this file with any web browser, and will be prompted to enter a value for “Filter by Counts per OTU”. Only OTUs with total counts at or above this threshold will be displayed. The OTU heatmap displays raw OTU counts per sample, where the counts are coloured based on the contribution of each OTU to the total OTU count present in that sample.

```
make_otu_heatmap_html.py -i out/otu_table.biom -o out/otu_heatmap
```

Taxonomy Summary Charts

The taxa of the samples can be visualised at each taxonomic level (see the **–L** flag).
Here**, summarize_taxa.py** produces a text file at the Phylum level (Level 2=Domain, 3=Phylum, 4=Class, 5=Order, 6=Family, 7=Genus) and **plot_taxa_summary.py** produces the html output.

```
summarize_taxa.py -i out/otu_table.biom -o out/taxa_summary -L 3

plot_taxa_summary.py  -i out/taxa_summary/otu_table_L3.txt -l Phylum -o out/taxa_charts -k white
```

Diversity

Community ecologists typically describe the microbial diversity within their study. This diversity can be assessed within a sample (alpha diversity) or between a collection of samples (beta diversity).

Alpha

Alpha diversity will be calculated and displayed though using this workflow. The full list of metrics available can be found at http://qiime.sourceforge.net/scripts/alpha_diversity_metrics.html. The html visualisation file can be found at ‘out/arare/alpha_rarefaction_plots/rarefaction_plots.html’

```
alpha_rarefaction.py -i out/otu_table.biom -m Fasting_Map.txt  -o out/arare  -p custom_parameters.txt  -t out/rep_set.tre
```

Beta

Beta diversity can be represented in many different ways, shown below. By rarefying the samples to the smallest set (in this example dataset, 146 sequences) sample heterogeneity can be removed.
Firstly, 3d plots are generated using unifrac.

```
beta_diversity_through_plots.py -i out/otu_table.biom  -o out/bdiv_even146  -p custom_parameters.txt
        -m Fasting_Map.txt  -t out/rep_set.tre -e 146
```

To view a 3d plot, navigate to the jar directory within the metric you wish to view (weighted/unweighted, continuous/discrete) and enter ‘java -jar jar/king.jar */*.kin’ where you can then view the output. The more traditional 2d plots are also generated by unifrac:

```
make_2d_plots.py -i out/bdiv_even146/unweighted_unifrac_pc.txt  -o out/bdiv_even146/unweighted_unifrac_2d
            -m Fasting_Map.txt  -k white -p out/bdiv_even146/prefs.txt
```

These are easiest viewed through the html page:
'out/bdiv_even146/unweighted_unifrac_2d/unweighted_unifrac_pc_2D_PCoA_plots.html'

Inter-Sample Distance

Distance Histograms are a way to compare different categories and see which tend to have larger/smaller distances than others.

```
make_distance_histograms.py -d out/bdiv_even146/unweighted_unifrac_dm.txt
            -m Fasting_Map.txt -o out/bdiv_even146/distance_histograms -p out/bdiv_even146/prefs.txt
```

The html is found at:
'out/bdiv_even146/distance_histograms/unweighted_unifrac_dm_distance_histograms.html'

Jackknifing & UPGMA

To measure robustness of the sequencing effort, we perform a jackknifing analysis, wherein a small number of sequences are chosen at random from each sample, and the resulting UPGMA tree from this subset of data is compared with the tree representing the entire available data set. This produces jackknifed weighted and unweighted 2d and 3d plots like above, and also jackknifed trees found in the **out/jack/** directory.

```
jackknifed_beta_diversity.py -i out/otu_table.biom -o out/jack -p custom_parameters.txt
             -e 110 -t out/rep_set.tre -m Fasting_Map.txt

make_bootstrapped_tree.py -m out/jack/unweighted_unifrac/upgma_cmp/master_tree.tre -s
            out/jack/unweighted_unifrac/upgma_cmp/jackknife_support.txt -o
            out/jack/unweighted_unifrac/upgma_cmp/jackknife_named_nodes.pdf

evince out/jack/unweighted_unifrac/upgma_cmp/jackknife_named_nodes.pdf
```

A key feature of the QIIME interface is the ability to list the steps which you wish to run and have them sequentially performed by running them as a standard shell script. In the file **qiime_tutorial_commands_serial.sh** in your working qiime directory, you will find the commands which we have just gone through. This can be called directly from the QIIME shell prompt and will produce the same output as we have achieved, with no user input. This can be edited, along with **custom_parameters.txt** to tune the analyses to your specific requirements.

What is described above is a brief introduction to the type of analyses which QIIME can perform. Extensive details of the commands, parameters and metrics used can be found at http://www.qiime.org/scripts *or through typing a QIIME command followed by* ***'-help'*** *into the qiime shell prompt.*

Analysing sequences with MOTHUR

MOTHUR is another popular pipeline for performing microbial community analysis that integrates many third party tools which have become standard in the field. MOTHUR is included in the standard Bio-Linux distribution.

As an example, we will use the same data used in the previous QIIME tutorial. Please refer to the previous QIIME tutorial for the description of the experiment and the data.

What follows is an adapted version of the exemplary tutorial provided by MOTHUR (which can be found at http://www.mothur.org/wiki/Sogin_data_analysis). Further details and parameters on the below commands and many more can be found at this site. The material was compiled and adapted by Soon Gweon, NBAF.

Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Schloss, P.D., et al., Appl Environ Microbiol, 2009. 75(23):7537-41*

Preparation

First, we must copy the tutorial data to your home directory and extract it:

```
cd
tar  -xvzf /usr/local/bioinf/documentation/bio-linux/intro_course/mothur_tutorial_data.tar.gz
cd mothur_tutorial_data
```

Entering the directory (cd mothur_tutorial_data) and listing the files (ls) will show what was extracted:

Fasting_Example.fna
This is the 454-machine generated FASTA file.
Fasting_Example.qual
This is the 454-machine generated quality score file, which contains a score for each base in each sequence included in the FASTA file.
Fasting_Example.oligos
This is generated by the user. This file is used to provide barcodes and primers to MOTHUR.

data
This directory contains the reference files required for alignment of the OTUs.

To begin working with MOTHUR, you must enter the MOTHUR shell by typing '**mothur**' in your working directory. This has been successful if the prompt changes to end in '**mothur >**'. The commands below will only be recognised within the special MOTHUR shell.

```
mothur
```

Assign Samples to Multiplex Reads and Quality Filtering

First, we need to separate each sequence according to the barcode and primer combination. The first task is to assign the multiplex reads to samples based on their nucleotide barcode using the information from oligos file. Also, this step screens sequences based on the quality file, truncating reads at where the quality score falls below the threshold. The script for this step is **trim.seqs**:

```
trim.seqs(fasta=Fasting_Example.fna, oligos=Fasting_Example.oligos, qfile=Fasting_Example.qual, qaverage=25,
minlength=200, maxlength=1000)
```

This creates five files in the current directory:

Fasting_Example.trim.fasta
This is the processed fasta file.
Fasting_Example.trim.qual
This is the precessed quality file.
Fasting_Example.scrap.fasta
This file contains sequences which fell below the thresholds (below quality score of 25, shorter than 200 bps or longer than 1000 bps)
Fasting_Example.scrap.qual
This is the quality file for the scrapped sequences.
Fasting_Example.groups
This is a two-column list with the first column indicating the sequence names of those sequences in the Fasting_Example.trim.fasta file and the second column the group that it came from.

Generating Alignment & Distance Matrix

The first thing we want to do is to simplify the dataset by working with only the unique sequences. We are not chucking anything here, we are just making the life of your CPU and RAM a bit easier. We do this with the command: **unique.seqs**

```
unique.seqs(fasta=Fasting_Example.trim.fasta)
```

We then need to generate an alignment of our data using the **align.seqs** command by aligning it to SILVA-compatible alignment database reference alignment. Please note that this step can take awhile to complete.

```
align.seqs(fasta=Fasting_Example.trim.unique.fasta, reference=data/silva.bacteria.fasta, flip=T)
```

Next, we need to filter our alignment so that all of our sequences only overlap in the same region and remove any columns in the alignment that don't contain data. We do this by running the **filter.seqs** command.

```
filter.seqs(fasta=Fasting_Example.trim.unique.align)
```

Next, we want to calculate the column-formatted distance matrix, but we are only interested in distances smaller than 0.15 at this stage. We will do this using **dist.seqs** command.

```
dist.seqs(fasta=Fasting_Example.trim.unique.filter.fasta, cutoff=0.15)
```

Classify Sequences

We then need to classify our sequences using the MOTHUR version of the “Bayesian” classifier. We do this with classify.seqs command using the SILVA-compatible reference file and taxonomy file (http://www.mothur.org/wiki/Silva_reference_alignment)

```
classify.seqs(fasta=Fasting_Example.trim.unique.filter.fasta, name=Fasting_Example.trim.names,
template=data/silva.bacteria.fasta, taxonomy=data/silva.bacteria.silva.tax)
```

Renaming Files

This step is done only to make our life easier by making copies of some files and giving it nice and short names. The command **system()** allows you to run programs outside of MOTHUR without leaving the MOTHUR shell.

```
system(cp Fasting_Example.trim.unique.filter.fasta final.fasta)
system(cp Fasting_Example.trim.names final.names)
system(cp Fasting_Example.groups final.groups)
system(cp Fasting_Example.trim.unique.filter.dist final.dist)
system(cp Fasting_Example.trim.unique.filter.silva.wang..taxonomy final.taxonomy)
```

Clustering Sequences

Now we want to assign these sequences to OTUs for every possible distance up to and including a distance of 0.15. By default, this method uses the average neighbour algorithm.

```
cluster(column=final.dist, name=final.names, cutoff=0.15)
```

Generating OTU Table and Normalisation

Now that we have a list file, we need to create a table that indicates the number of times an OTU shows up in each sample. This is called a shared file and can be created using the **make.shared** command. We are only interested in the distance of 0.03 from the list file, so we give 0.03 to “label” parameter.

```
make.shared(list=final.an.list, group=final.groups, label=0.03)
```

We then normalise the number of sequences in each sample. In order to do this, we need to know how many sequences are in each step. You can do this with the **count.groups** command.

```
count.groups()
```

From the output we see that the sample with the fewest sequences had 146 sequences in it, so we normalise all the samples to this number of sequences.

```
sub.sample(shared=final.an.shared, size=146)
```

Classifying OTU

The last thing we'd like to do is to get the taxonomy information for each of our OTUs. To do this we will use the **classify.otu** command to give us the majority consensus taxonomy.

```
classify.otu(list=final.an.list, name=final.names, taxonomy=final.taxonomy)
```

Converting the shared file to BIOM-format

The **make.biom** command allows you to convert your shared file to a biom file. Please refer to http://biom-format.org/documentation/biom_format.html for detail.

```
make.biom(shared=final.an.shared, contaxonomy=final.an.unique.cons.taxonomy)
```

Data to information

MOTHUR has many different ways to visualise and interrogate the data. Here we will explore just a few.

Heatmap

Now we'd like to compare the membership and structure of the various samples using an OTU-based approach. Let's start by generating a heatmap of the relative abundance of each OTU across the 24 samples using the heatmap.bin command.

```
heatmap.bin(shared=final.an.shared)
```

The output will be in a SVG-formatted file called final.an.0.03.heatmap.bin.svg. In this heatmap, the red colors indicate communities that are more similar than those with black colors.

Venn Diagram

MOTHUR allows you to generate a Venn diagram with **venn** command. Let's take a look at the Venn diagram for PC.354 and PC.355.

```
venn(shared=final.an.shared, groups=PC.354-PC.355)
```

This generates a file called final.an.0.03.sharedsobs.PC.354-PC.355.svg. To view the file, type the following in **another terminal**:

```
eog final.an.0.03.sharedsobs.PC.354-PC.355.svg
```

When generating Venn diagrams we are limited by the number of samples that we can analyze simultaneously. MOTHUR can generate up to 4-way Venn diagram:

```
venn(shared=final.an.shared, groups=PC.354-PC.355-PC.356-PC.481)
```

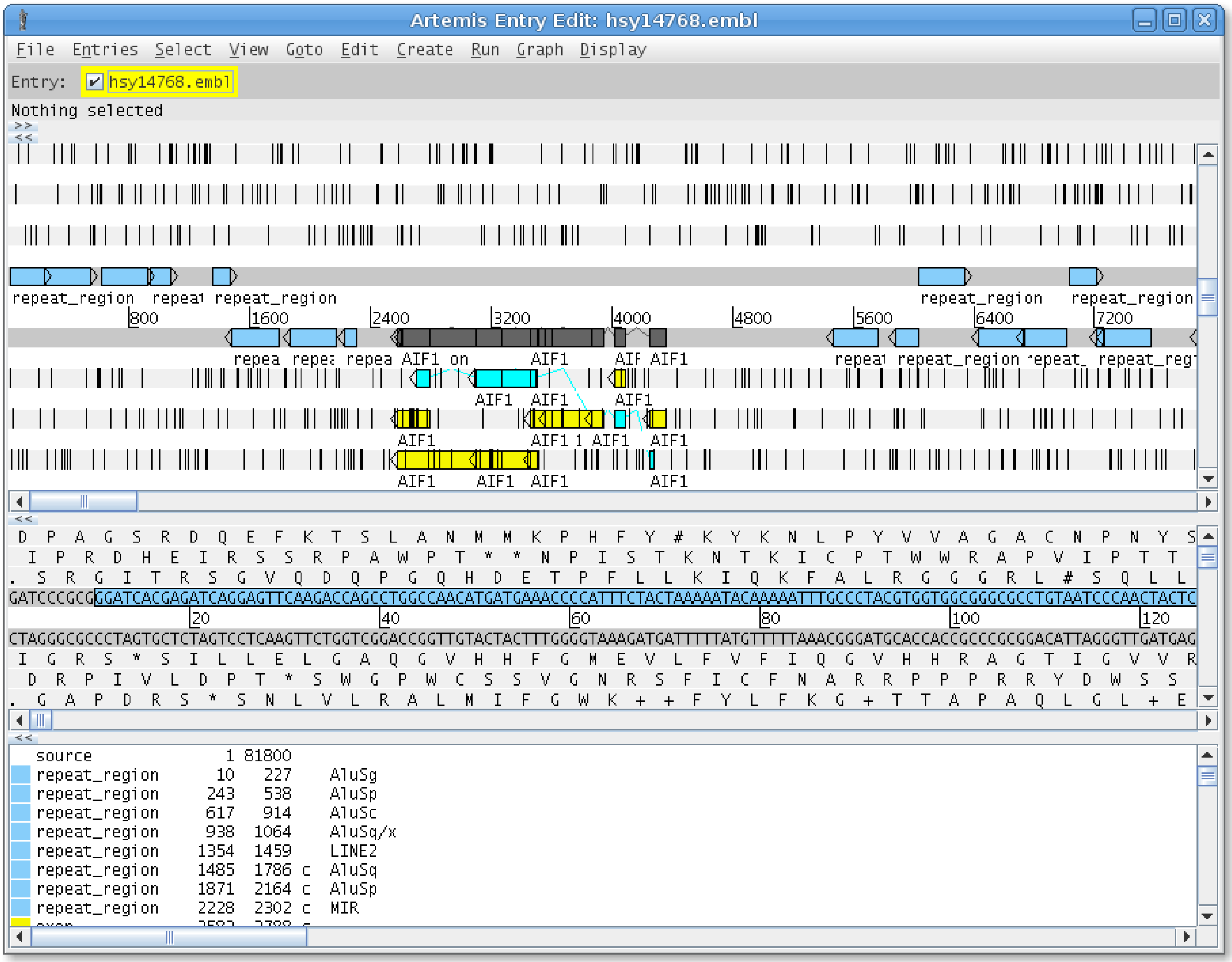
Artemis

Artemis is a DNA sequence viewer and annotation tool, allowing visualisation of sequence features and the results of analyses within the context of the sequence and its six-frame translation. Artemis can read embl or genbank format files. Sequences can be loaded from local files or via the network from the EBI.

Ways to run Artemis:

- from a locally installed version on your Bio-Linux machine*
- via Java Web Start from the Sanger Centre

(http://www.sanger.ac.uk/resources/software/artemis/java/artemis.jnlp)



```
 D   P   A   G   S   R   D   Q   E   F   K   T   S   L   A   N   M   M   K   P   H   F   Y   #   K   Y   K   N   L   P   Y   V   V   A   G   A   C   N   P   N   Y   S
   I   P   R   D   H   E   I   R   S   S   R   P   A   W   P   T   *   *   N   P   I   S   T   K   N   T   K   I   C   P   T   W   W   R   A   P   V   I   P   T   T
.  S   R   G   I   T   R   S   G   V   Q   D   Q   P   G   Q   H   D   E   T   P   F   L   L   K   I   Q   K   F   A   L   R   G   G   G   R   L   #   S   Q   L   L
GATCCCGCGGGATCACGAGATCAGGAGTTCAAGACCAGCCTGGCCAACATGATGAAACCCCATTTCTACTAAAAATACAAAAATTTGCCCTACGTGGTGGCGGGCGCCTGTAATCCCAACTACTC
                    20                  40                  60                  80                  100                 120
CTAGGGCGCCCTAGTGCTCTAGTCCTCAAGTTCTGGTCGGACCGGTTGTACTACTTTGGGGTAAAGATGATTTTTATGTTTTTAAACGGGATGCACCACCGCCCGCGGACATTAGGGTTGATGAG
 I   G   R   S   *   S   I   L   L   E   L   G   A   Q   G   V   H   H   F   G   M   E   V   L   F   V   F   I   Q   G   V   H   H   R   A   G   T   I   G   V   V   R
   D   R   P   I   V   L   D   P   T   *   S   W   G   P   W   C   S   S   V   G   N   R   S   F   I   C   F   N   A   R   R   P   P   P   R   R   Y   D   W   S   S
.  G   A   P   D   R   S   *   S   N   L   V   L   R   A   L   M   I   F   G   W   K   +   +   F   Y   L   F   K   G   +   T   T   A   P   A   Q   L   G   L   +   E
```

```
  source            1 81800
  repeat_region    10   227    AluSg
  repeat_region   243   538    AluSp
  repeat_region   617   914    AluSc
  repeat_region   938  1064    AluSq/x
  repeat_region  1354  1459    LINE2
  repeat_region  1485  1786 c  AluSq
  repeat_region  1871  2164 c  AluSp
  repeat_region  2228  2302 c  MIR
  exon           2583  2798 c
```

Figure 14: Artemis Entry window after hsy14768.embl is loaded.

Exercise

- Start Artemis on Bio-Linux by typing **artemis** on the command line ***or*** by choosing the option **Artemis** from under the **Bioinformatics Applications** graphical menu.

- Now choose the option ***Open...*** from under the Artemis File menu, and select the file **hsy14768.embl** from within the bioinf_files directory.

This should open up a large window, as shown in Figure 14, where this sequence is displayed graphically .

- Open a terminal window and view the text of the embl entry using the command **less hsy14768.embl**

*Notice how **Artemis** is providing a graphical representation of what is in the text file.*

- Try choosing **Mark Open Reading Frames** from under the **Create** menu of Artemis.

- Choose to mark open reading frames with a minimum size of 200.

*You should now see two boxes near the top in the **Entry** section, the first called* **hsy14768.embl** *and the other called* **ORFS_200+**.

- Uncheck the box next to **hsy14768.embl**. You should now be able to scroll along the window horizontally and easily see the open reading frames you marked.

- Check the box next to **hsy14768.embl** again. Look at the information in the bottom frame of the window. Notice how it is related to the images in the frames above.

- Try clicking on some of the lines in the bottom frame and seeing what happens in the images in the other two frames.

- Explore the options available to you. (Not all options will be functional by default. See the information about the Run menu below)

- Close the Artemis Entry Editing window using **File | Close**.

- You can also load up files direct from the EBI. If you want to try this, then choose **File | Open from the EBI – Dbfetch...** option in the original small Artemis window and enter the accession number **BX255937**.

- **When you are done, close Artemis by choosing File | Close in the sequence entry window and then choosing File | Quit in the main (small) Artemis window.**

You can run various programs on your sequence, or parts of your sequence, from under the **Run menu** in Artemis. Some of the options in this menu need to be configured to be appropriate for your site. There is information on how to do this on our website at:

http://nebc.nerc.ac.uk/tools/bioinformatics-docs/faq#blast_art

If you are not the system administrator of your Bio-Linux machine, then you will probably need to liaise with the person who is to get this set up properly.

We also highly recommend ***Artemis***' sister program ***Act***, which can be used to graphically view a pairwise BLAST betrween two or more sequences.

Finding and running useful scripts

Exercise

http://nebc.nerc.ac.uk/tools/code-corner/scripts

- Visit the above link, then find the "fastagrep" script located under "Sequence Formatting and Other Text Manipulation".
- Make a folder called "scripts" in your home directory and save the file there.
- In a terminal run the command **chmod a+x scripts/fastagrep** to tell Linux that this file is an executable script.
- Type **scripts/fastagrep** to run the script. In this case you will see basic help.

Fastagrep is a script to help extracting sequences of interest form a multi-FASTA file by matching text in the header lines. It is a FASTA-aware version of the standard Linux 'grep' command introduced in part 1. The script will be used in the next exercise on sequence alignment.

Tip:

- If you get a "permission denied" error when running the script, it normally means that you missed out the **chmod a+x ...** part.
- If you get a "bad interpreter" error it means that the interpreter named on the first line of the file cannot be found on the system. You can always run the interpreter explicitly – eg. by typing **perl scripts/fastagrep**.

Aligning sequences using MUSCLE

Aligning multiple sequences is a very common task, as it is the first step to comparing related sequences. There are many algorithms for performing gapped global alignments over a set of sequences, most of which can be used on either nucleotide or peptide input. Many web based tools offer to align sequences, for example http://uniprot.org can align sequences retrieved from a search on the reference database, and additional sequences can also be uploaded and added to the alignment. GUI applications like ClustalX and Jalview can call alignment applications like Clustal, MUSCLE, and MAFFT for you and display the results graphically.

Sometimes you may want to run the alignment directly from the command line – reasons for this include:

- You want to fine tune the options passed to the aligner
- You want to use an aligner program that is not supported by the GUI or website you are using
- You want to run the alignment remotely – for example on a powerful departmental server
- You want to run several alignments at once using a loop or a short script

Exercise

Plants contain many closely related genes in the cellulose synthase family. Previous studies have examined these in some model organisms, eg maize[ref below]. It might be useful to compare the cellulose synthase genes in another plant of interest, or to align bacterial homologues against the plant genes.
For use in this exercise, the file **all_cellulose_synthase.fasta** in the example files directory contains all the reference cellulose synthase genes from Uniprot (selected with the query “name:cellulose synthase”).

1. Ensure that you have the fastagrep script available from the previous exercise.
2. Use 'fastagrep' to extract all the sequences that come from oilseed rape (Brassica napus):

   ```
   scripts/fastagrep -F 'OS=Brassica napus' bioinf_files/all_cellulose_synthase.fasta
   ```

   - Here, the -F flag specifies an exact text match and the 'OS=...' syntax is specific to the headers returned by Uniprot.
3. This will print the matching sequences to the terminal. To save them, use file redirection:

   ```
   scripts/fastagrep -F 'OS=Brassica napus'   \
       bioinf_files/all_cellulose_synthase.fasta > seqs.fasta
   ```

4. Now we can invoke MUSCLE with the default parameters:

   ```
   muscle -in seqs.fasta -out seqs.aln
   ```

5. Run the Jalview application from the bioinformatics menu. Close the default project windows that appear, and select “Input Alignment -> from File”. Now load seqs.aln, enable colouring in the Colour menu and bring up the overview window from the view menu.

Jalview has many options for viewing and editing the alignment, drawing trees, etc.

For comparing alignments, you may want to add the “-stable” flag to the muscle command in order to maintain the sequences in the same order as the input FASTA file.

[ref for paper mentioned above]
Holland et al. 2000. A comparative analysis of the plant cellulose synthase (CesA) gene family.
http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&cmd=search&term=10938350

BLAST

The Basic Local Alignment Search Tool (BLAST) searches for regions of **local** similarity between sequences. The program compares nucleotide or protein sequences or patterns to sequence, or sequence-related, databases and calculates the statistical significance of matches.

The documentation here covers only the most commonly used BLAST implementation, BLAST+ from NCBI. There are several other BLAST varients that essentially do the same thing. Some are commercial, for example AB-BLAST from Advanced Biocomputing LLC, formerly known as WU-BLAST. There are also many other programs that search sequence databases and perform local alignments. Before relying on BLAST as your search tool you should consider whether one of these might better suit your analysis needs.

A few examples of ways to run BLAST, on Bio-Linux or otherwise

- Locally installed command line against locally installed BLAST databases
- Locally installed command line against remote databases
- Locally through options in graphical programs (e.g. under the Run menu in Artemis)
- Remotely through ssh tunnelling or the remote BLAST options in Artemis.
- Remotely on websites such as those available at the NCBI and EBI
- Remotely using webservices, either through programs such as Taverna, or through scripting

For this course, we assume that you are familiar with running BLAST searches using at least one web-based interface. If you are not, then this is a good time to look at the facilities offered through one of these sites, and to try BLASTing some of the example sequences in the coruse folder:

NCBI:	**http://blast.ncbi.nlm.nih.gov/Blast.cgi**
EBI:	**http://www.ebi.ac.uk/Tools/sss/**

Bio-Linux includes both the BLAST+ package and the older NCBI “blastall” implementation. Information and links in the Bio-Linux Bionformatics Documentation System (icon on your Desktop) provide information on both packages. The ncbi-blast+ package contains a number of programs allowing you to carry out different types of searches, as well as to create databases, reformat reports, etc.

What this course covers

This course covers how to run BLAST+ programs via the command line and a few simple steps you can take to work with more than one sequence at a time. We also cover how to install your own BLAST databases in Appendix C. We do not cover the internals of BLAST searching in any detail or how to interpret BLAST results.

Why use BLAST on the command line?

The web resources available for BLAST are highly developed, usually stable, and have access to a much greater set of data than most people will have available locally. They also often provide lovely graphics and links out to other data resources or analysis programs. So why use the command line at all?

For small volumes of data, where you wish to search a commonly available database or subset of data available through a website, then web access is a very good option. Web-based utilities are also good for experimenting with parameters when determining useful settings for your investigation. The command line comes into its own for setting up searches quickly, for processing large volumes of data, for automating your searches, and for giving you the ability to get just the information you want returned from the BLAST searches. (This last point has been made easier than ever in the newer BLAST+ programs, where you can, to a certain extent, specify which information to return in a tab delimited format[1].)

[1] You can return most information you want using the tab delimited output options in BLAST+. However, a key thing missing is the Description field – usually the most interesting field for a biologist! To get this field, along with others, out of a BLAST report, it is still necessary to consider custom scripting – or grabbing someone else's script that does the job!

We ***HIGHLY*** recommend you invest time learning about what BLAST does in detail, including how it works and what the statistics is produces mean. The "take the top hit" method will rarely serve your research well.

We provide a list of references and helpful web pages in **Appendix C** that we hope will help you learn more about blast programs.

General considerations for database searching

Database searching should be approached like an experiment. In particular: define your aims before your start. This will save you an enormous amount of time, both in terms of time taken doing searches and time taken bringing together and reporting your findings later.

Before you start searching with a sequence, it is useful to outline your answers to questions like:

- What am I trying to find out/what do I want to do with the results?
- What kind of database do I want to search with my sequence? E.g. nucleotide, protein, pattern, profile?
- Which database(s) in particular do I want to search? Why?
- Are there are any subsets of the database that I could or should restrict my search to?
- Do I want to take into account potential frameshifts in my coding sequences?
- What format is my sequence in?
- Do I want to filter my sequence for repeats and low complexity regions before searching?
- Is the scoring system I've chosen appropriate?
- Where and how will I store a record of the parameters I've used and the database version I've searched with?

A very, very brief introduction to BLAST+

BLAST+ includes programs to perform searches with different types of input against databases holding different types of data. Each search combination is referred to by a particular name and has its own command. A table of the basic BLAST "flavours" and what they do is given below.

Blastall flavour	Input sequence type	Database sequence type
blastn	nucleotide	nucleotide
blastp	peptide	peptide
blastx	nucleotide (6 frame conceptual translation is created during run)	peptide
tblastn	peptide	nucleotide (6 frame conceptual translation is created during run)
tblastx	nucleotide (6 frame conceptual translation is created during run)	nucleotide (6 frame conceptual translation is created during run)

There are many other programs available as part of the BLAST+ release apart from the ones above. These include **blastdbcmd, dustmasker, psiblast, rpsblast+, segmasker** and **srsearch.**. These programs are not covered here, but are worth learning about for your own work.

How a BLAST database looks on the file system

A typical BLAST database consists of three files names with extensions **.pin .phr and .psq** for protein databases or **.nin .nhr and .nsq** for nucleotide databases. These files represent a specially indexed version of a multi-fasta source file. Do not try to examine the files in a regular text editor (they appear as garbage), and do not try to split the files apart. When invoking BLAST commands, just give the path to the database without any extension (see examples). BLAST will know to find and read the three files.

A simple blastp search

The following is a basic blastp command – you can run it from within the course folder.

```
blastp -db blastdb/sprot –query cd4_cerae.fasta –evalue 0.0001 > cd4_cerae.blastp
```

The command is easy to understand when you break it down. It means:

- ➔ **run blastp**, i.e. a peptide sequence will be used to search a peptide database.
- ➔ The **database (-db)** to be searched is called **sprot** and can be found in the **blastdb** directory.
- ➔ The **input sequence (-query)** is **cd4_cerae.fasta**.
- ➔ Only report results of sequences **with e-values (-evalue)** better than (i.e. lower than) **0.0001**.
- ➔ Put the **results of this search** in the file **cd4_cerae.blastp**, using standard shell redirection **(>)**.

You can fine tune BLAST easily using additional command line options. We ***highly recommend*** that you read about BLAST and determine appropriate settings for your research questions. This will ultimately save you a huge amount of time and energy.

A copy of the Swissprot part of Uniprot, formatted for BLAST searches, is located in the directory **blastdb**, under your **bioinf_files** directory. We do not fully cover the use of **makeblastdb** in this course, but some more info is shown in Appendix C. For completeness, the steps we took, including the command we used to create the BLAST formatted Swissprot database, are as follows:

We downloaded the fasta formatted swissprot file from

ftp://ftp.ebi.ac.uk/pub/databases/fastafiles/uniprot/swissprot.gz

into the blastdb directory under bioinf_files.

We then used the **makeblastdb** command in a one-liner run within the blastdb/ directory.

```
gunzip -c swissprot.gz | makeblastdb -title Swissprot -out sprot -dbtype prot -in -
```

Note the use of a hyphen "-" in place of a filename tells the command to get the input via the pipe "|". This does not work in all cases but is a common convention in command line tools.

Reference databases for BLASTing would normally be stored in a shared location

You can either give the full or relative PATH to your blast databases within the blast command, or you can store your blast databases in a location that is supplied as the value for the BLASTDB environmental variable and just provide the database name in the blast command line.

When loading reference BLAST databases onto Bio-Linux 6 you can can put them in the default BLASTDB location **/home/db/blastdb** OR change the environmental variable **BLASTDB** to a location appropriate for your work. If you do not have **sudo** access you will need to talk to the system administrator of the machine about this. *Note that the default location for blast databases may be different on different machines, and may change on Bio-Linux in the future.*

For the purposes of this tutorial, we will give each BLAST command the explicit location of the BLAST database to search.

Exercise

- Move into the **bioinf_files** directory if you are not already there.

- List the files in the **blastdb** subdirectory. The files called sprot.p* are the files that BLAST uses when it searches.

- From within the **bioinf_files** directory, run the example command given previously, ie:

blastp -db blastdb/sprot –query cd4_cerae.fasta –evalue 0.0001 > cd4_cerae.blastp

- Look at the results file that has been created.

- Try a **blastx** search on the file unknown.fasta. This time set the evalue to 1 and save the results in unknown.blastx. The command you use will start like this:

blastx -db blastdb/sprot -query unknown.fasta ...???...

Recall that a **blastx** *search translates a nucleotide sequence in six frames and searches a peptide database.*

- Look at the results file.

- **blastp** expects a peptide query file, and **blastx** expects nucleotides. What would you expect to happen if you use an inappropriate BLAST flavour? Try it and see.

Formatting BLAST output

You have now seen the default report format for BLAST searches. There are many options available using the **-outfmt** option with a numerical argument between 0 and 11. The default is **-outfmt 0**.

The BLAST+ commands don't (currently) have man pages, but to see a list of all the **-outfmt** options you can use the builtin help function:

blastx -help | less

Exercise

- Run either of the above BLAST searches again, this time adding the parameter **-outfmt 6** to the command. Make sure you change the name of the output file as well, or else just let the results get printed to the screen.

- Look at the results from this search and compare it to what was returned using default formatting. Is it easier or harder to read? Is there information present in one report that is not in the other?

Note: *BLAST+ programs offer finer control over the format and contents of results returned – see the help page as mentioned above.*

Handling multiple sequences

This section covers ways to deal with a small number of sequences at once – say up to a few hundred. For thousands of sequences, you will probably want to use the ideas introduced here, in conjunction with running your searches on a cluster and using scripts for pulling out the information of relevance to you from the results files.

This section looks at using BLAST on a number of sequences. However, **the general principles presented apply to many bioinformatics programs**.

The two methods covered here are:

1) Using files containing more than one sequence
2) Using multiple sequences as input using a "foreach" loop

BLAST searching using fasta files containing more than one sequence

Exercise

- Look at the contents of the file **multiseqs.fasta** in your **bioinf_files** directory. How many sequences are in this file?

- Run a blastx search using multiseqs.fasta as the input file.

```
blastx -db blastdb/sprot -query multiseqs.fasta -evalue 0.4 > multiseqs_1.blastx
```

- Look at the results file to see how the results have been reported. How easy would this be to read and understand? Could you load the results into other software tools?

- Try the above query again, but with the **-outfmt 6** flag.

- Read about the **-num_descriptions, -num_alignments and -max_target_seqs** flags in the BLAST+ documentation. For very small studies, where you might read through the BLAST reports yourself rather than doing further processing on them using the computer, these flags may help you otherwise.

BLAST searching using a foreach loop

A foreach loops say to the computer:

"For each thing in this list, do the following:"

So, when running multiple BLAST searches, you might want to do something like:

"For each sequence in my list, run a blastx search against my Swissprot database."

You can also create nested foreach loops. For example, if you had a list of sequences and a list of databases, you could use a nested foreach loop to get the computer to do something like this:

"For each sequence in my sequence list, run a blastx search against each database in my database list"

You can run a foreach loop on arbitrarily long lists. However, for the exercises below, we will use just five sequences:

testseq1.fasta, **testseq2.fasta**, **testseq3.fasta**, **testseq4.fasta** and **testseq5.fasta**.

The foreach loop explained step by step

Please note that the syntax used this section assumes that you are in the default Z-shell. If the commands fails for you and you are sure that you have typed them in correctly, please check your shell.

You can identify your current shell by typing the command **echo $0**. If you are not in the z-shell (zsh) already, just type **zsh** in your terminal window.

Other shells provide the same functionality as the foreach loop demonstrated here, but the syntax is different.

You need to tell the computer the list of files to work on. Here, we will use a glob pattern match to indicate the list of sequences we want to work with. Recall that **echo** simply prints its arguments and so can be used to show glob expensions:

```
echo testseq*.fasta
```

or, if we wanted to be more specific:

```
echo testseq[1-5].fasta
```

We bind each file in the list to a *loop variable* within the first line of the foreach loop. So the following says: "take each file in this list in turn and refer to it as **j**":

```
foreach j in testseq[1-5].fasta
```

When we finish, our complete foreach loop will state:

```
foreach j in  testseq[1-5].fasta ; do
blastx –db blastdb/sprot  -query  $j  -evalue 0.01  -out $j.blastx
done
```

This means: *for each sequence in the list in the first line, run the command in the second line. When all the sequences in the list have been dealt with, then finish.*

Loops are very powerful and useful, so it is worth understanding exactly how they work. A more detailed explanation follows.

Explanation of the first line of a foreach loop:

- we have used the command "**foreach**". It's not the only way to write a loop but it is the most used.

- the "**j**" is a name we choose to refer to "**each thing**" – more specifically, for *each thing* we get to in the list, let's refer to it by the name **j**. This is an arbitrary name. You can use whatever you want. So the following are equally correct to the line given above:

foreach myThing in testseq[1-5].fasta	*calls each list item in turn "**myThing**"*
foreach x in testseq[1-5].fasta	*calls each list item in turn "**x**"*
foreach seq in testseq[1-5].fasta	*calls each list item in turn "**seq**"*

Once you have chosen a name for *each thing* in your list, you must use that name with a dollar symbol "$" to refer to the list item in any commands that follow within the foreach loop. Recall how the $ construct also lets you access the contents of environment variables, like $BLASTDB.

- The keyword **in** is followed by a list of things to loop over. In this case the list is being generated as the result of a single glob pattern expansion, but this need not be the case. You can list items explicitly, use multiple patterns, or even generate a list on-the-fly using backtick substitution (not covered in this tutorial).

- The semicolon serves to terminate the list of items to be processed, and **do** primes the shell to accept one or more commands to be run within the loop. The single command **done** terminates this list.

- So the overall effect of that one line is*: "foreach thing that matches the pattern **testseq[1-5].fasta**, do the following:",* and after that you just supply a regular command to run. Note how we can reference **$j** as the input sequence and also use **$j.blastx** to generate a filename for the results – ie. the original name with .blastx appended.

Hint: It is usually a good idea to check that the command or pattern used to create a list does actually generate the list you expect before including it within a foreach loop. Once common trick is to add **echo** on the start of the command within the loop, so the commands are printed to the screen but not run.

Exercise

We will set up a foreach loop to run blastx searches using the five testseq*.fasta sequences with the Swissprot database.

- Type this command to begin the foreach loop as described above:

  ```
  foreach j  in testseq[1-5].fasta ; do
  ```

- You will now be seeing something like:

  ```
  live@machine[bioinf_files] foreach j in  testseq[1-5].fasta ; do
  foreach>
  ```

- The **foreach>** is a prompt, much like the regular prompt – it is here we tell the computer what we want it to do with each item in the list. To do this, type:

  ```
  blastx –db blastdb/sprot  -query  $j  -evalue 0.01  -out $j.blastx
  ```

  Recall that we defined *each thing* that we want to work on by the letter **j** in the first line of the foreach loop. In each subsequent line of the foreach loop, we refer to *each thing* by prefacing the **j** with a **$** sign.

  *Each **$j** in that command will be replaced by the name of a file from the list.*

  So here, the blastall command is executed with each filename in turn, and output files are named using the sequence filename with **.blastx** appended.

- You will now see another **foreach>** prompt, inviting a second command, but you are done so type

  ```
  done
  ```

  This indicates that there are no more processing steps to include in this foreach loop.

  If when running the above you get an error such as “File testseq[1-5].fasta not found”, it means the pattern did not match any file. Try simply running **echo testseq[1-5].fasta** to see what files the pattern is matching in the current directory.

- After running the foreach loop successfully, type the command

  ```
  ls  -l  *blastx
  ```

You should now see that you have five blastx results files. Imagine you had 100 sequences to blast – you could set up a foreach loop and go get a coffee. (Of course, you still need to figure out how you're going to use or analyses the results files if you're working with large numbers of sequences.)

We mentioned above that the **j** in the foreach loop was an arbitrary name. As an example, if we had used **seq** instead of **j**, the foreach loop would have been written:

```
foreach seq  in  testseq[1-5].fasta ; do
blastx –db blastdb/sprot  -query  $seq  -evalue 0.01  -out $seq.blastx
done
```

Notice that we have just replaced each instance of **$j** with **$seq.** Be careful, as the shell will not notice if your names do not match up, but will just substitute blank spaces into the command.

Exercise

- Read through all the files called testseq*.blastx by using the command **less**:

  less testseq*.blastx

- When you get to the end of one document, (or just want to go to the next document), just type **:n**
- To quit, type **q**

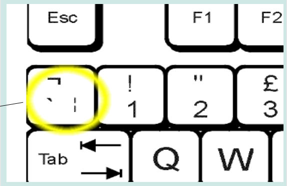
Why go to all this trouble when we could just create a multiple fasta file and run a BLAST search using it?

Foreach loops can be used with any programs – not just BLAST. So this method is widely applicable.

Multiple tasks, and even inner loops can be carried out in a single foreach loop, as the following example shows.

Exercise

If you have time, you can run the following foreach loop. Try to figure out what it does before running it. You may need to read the man pages for **basename** and **cut** to understand all the steps being taken. Note, the text has been indented for clarity but you need not type it like this. Also note the special quotes in the second line are **backticks** obtained with the key at the top left of the keyboard, next to number 1. These serve to *capture* the output of the **basename** command into the **newname** variable, and later to drive an inner loop from a list contained in a file. (Earlier, we said these wouldn't be covered in the course, but here's a little taster. Backticks are a powerful feature for any aspiring command-line guru to master!)



```
foreach seq in  testseq[1-3].fasta ; do
  newname=`basename $seq  .fasta`
  mkdir $newname
  pushd $newname
  blastx -db ../blastdb/sprot  -query  ../$seq  -evalue 0.01 -outfmt 6  -out $newname.blastx
  cat $newname.blastx | cut  -f2  >  top5.list
  for hit in `cat top5.list` ; do
    wget -q "http://www.uniprot.org/uniprot/$hit.txt"
  done
  popd
done
```

You can get the Z-shell to report what it is doing within loops and functions by runnign the command **set -x**. To return to normal output type **set +x.**

Working with lots of BLAST results

Reading a few BLAST reports is fine, but when you have thousands, you presumably won't be reading them one by one yourself.

A common way to handle large volumes of BLAST results is to get the computer to process the report files, pulling out key information. You can try using the various **-outfmt** options, which give you a great deal of fine tuned control over what to report in tab delimited format. Alternatively, you can use a customised script. You might choose to load such extracted information into a database, or for small scale studies, into a spreadsheet. This topic is not covered further in this course, but we recommend BioPerl modules for parsing BLAST report files. Example BioPerl scripts for BLAST parsing can be found on your Bio-Linux machine under the following directory:

/usr/share/doc/bioperl/examples/searchio

EMBOSS Programs

EMBOSS is an extensive package of programs that cover areas of bioinformatics analysis including:

- Sequence alignment
- Rapid database searching with sequence patterns
- Protein motif identification, including domain analysis
- Nucleotide sequence pattern analysis---for example to identify CpG islands or repeats
- Codon usage analysis for small genomes
- Rapid identification of sequence patterns in large scale sequence sets
- Presentation tools for publication

We recommend that you refer to the official EMBOSS overview at **http://emboss.sourceforge.net/what/#Overview** to find out more about the extensive functionality available via EMBOSS programs.

EMBOSS also consists of an underlying programming library, in case you are interested in building your own EMBOSS tools.

Ways to run EMBOSS programs:

- Locally installed, via the jemboss graphical interface on your Bio-Linux machine*
- Locall installed via graphical interfaces available under the Applications | Bioinformatics | Emboss menu
- Locally installed, via the command line on your Bio-Linux machine*
- Remotely on websites such as Mobyl: http://mobyle.pasteur.fr
- Remotely using webservices

Biological databases and EMBOSS on Bio-Linux

Certain EMBOSS programs can talk to local or remote biological databases. The version of EMBOSS installed on Bio-Linux machines is pre-configured to access data from embl, emblcds, uniprot (including swissprot and trembl) and Refseq from the EBI. Information about how to change this configuration can be found at

http://nebc.nerc.ac.uk/tools/bioinformatics-docs/other-bioinf/emboss-applications-and-databases

Sequence formats and EMBOSS

EMBOSS programs accept most common sequence formats. EMBOSS also includes a versatile tool called **seqret** that can be used to convert between sequence formats should you need to do this for other bioinformatics programs.

A comparison of the Jemboss and command line interfaces for EMBOSS programs

Interface	*Pros*	*Cons*
Jemboss *Graphical Interface*	Easy to see the programs available and what type of analysis they do Easy to run	Much slower to set programs running than on the command line Not always obvious how to save and where to save

	Many programs accept input files with multiple sequences, either directly or using lists of sequence or filenames. Documentation is easy to access	output Additional programs with EMBOSS interfaces are not available via this interface. e.g. there are emboss interfaces for phylip and hmmer programs, among others, which are useful when creating pipelines and automating tasks. Programs that are interfaces to others (e.g. emma is an EMBOSS interface to clustalw) may not always work smoothly via Jemboss, even though they are fine via the command line.
Command Line	Prompted command line makes programs easy to run Programs accept input files with multiple sequences either directly or using lists of sequence or filenames. Easy to automate tasks Easy to create pipelines of tasks Documentation easy to access	Prompted command line makes it easy to overlook many of the options available You have to read the documentation to find out about the options available

Working with EMBOSS programs

We will run a simple 3 stage task twice – once using Jemboss and once using the command line so that you can experience ,and get a feeling for the differences between, the two interfaces.

The task:
Fetch a sequence file from the EMBL database, extract all the mRNA sequences from the feature table and search for palindromes in those mRNA sequences.

Using Jemboss

Exercise

- Start Jemboss on Bio-Linux by typing **jemboss** on the command line. It can also be started by clicking on the icon under the **Applications | Bioinformatics** menu.
- Click on each of the categories (e.g. Alignment, Display, etc) to see what programs are listed.
- When you're finished exploring, click on the **Data Retrieval** category and choose **coderet** which is under **Sequence Data.**
- Scroll to the bottom of the window and click on the ⓘ button to bring up a documentation window. Read about what **coderet** does.

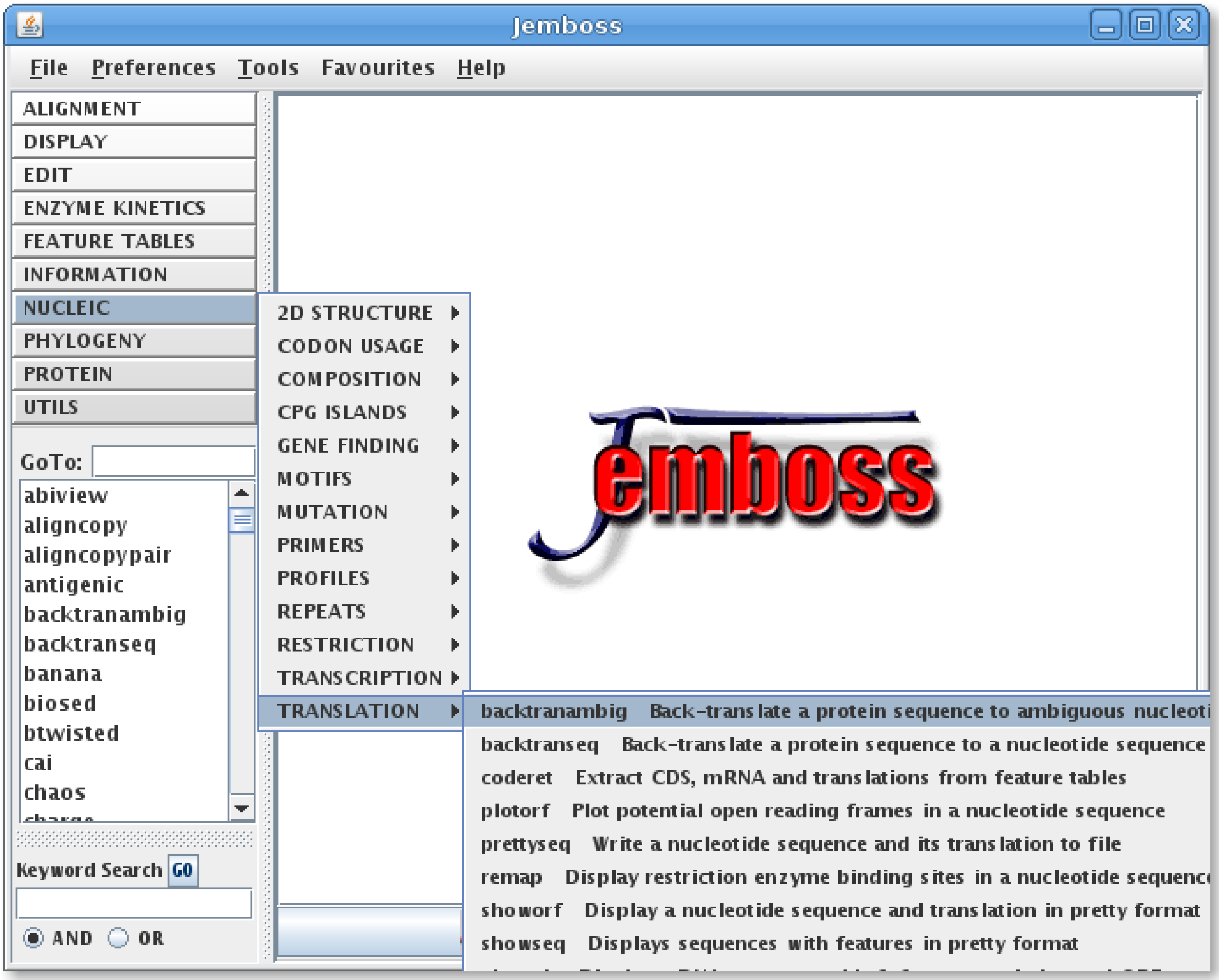

Jemboss
File
Preferences
Tools
Favourites
Help
ALIGNMENT
DISPLAY
EDIT
ENZYME KINETICS
FEATURE TABLES
INFORMATION
NUCLEIC
PHYLOGENY
PROTEIN
UTILS
GoTo:
abiview
aligncopy
aligncopypair
antigenic
backtranambig
backtranseq
banana
biosed
btwisted
cai
chaos
Keyword Search
GO
● AND
○ OR
2D STRUCTURE
CODON USAGE
COMPOSITION
CPG ISLANDS
GENE FINDING
MOTIFS
MUTATION
PRIMERS
PROFILES
REPEATS
RESTRICTION
TRANSCRIPTION
TRANSLATION
backtranambig Back-translate a protein sequence to ambiguous nucleoti
backtranseq Back-translate a protein sequence to a nucleotide sequence
coderet Extract CDS, mRNA and translations from feature tables
plotorf Plot potential open reading frames in a nucleotide sequence
prettyseq Write a nucleotide sequence and its translation to file
remap Display restriction enzyme binding sites in a nucleotide sequenc
showorf Display a nucleotide sequence and translation in pretty format
showseq Displays sequences with features in pretty format
Jemboss

Figure 15: The Jemboss graphical interface to EMBOSS programs

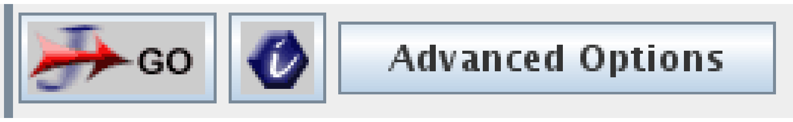


Figure 16: The **GO** button is pressed when you are ready to run the program. The *i* button pops up a window with documentation. Some, but not all programs, will also have an **Advanced Options** button that will bring up, often very useful, optional fields.

Exercise continued

- Scroll back to the top of the **coderet** form in the Jemboss window, and fill in a **Sequence Filename**. In fact, we want to pull a sequence directly from embl at the EBI. The sequence we want is from a plasmid and has the accession number U80928. To fetch it from the EBI, you need to type:

embl:U80928

into the **Sequence Filename** box.

- Enter a filename into the **outfile file name** box. For example, to distinguish from your later work, you could use the name: ***jemboss_bx.coderet***.

- Scroll to the bottom of the window and hit the **GO** button.

- When the program has finished, a new window called **Saved Results** should appear. (Don't be fooled – your results haven't been saved yet!) There should be a number of tabs in that window. One will be called the name you entered into the the **outfile file name** box (e.g. *jemboss_bx.coderet)* The others will likely be called things like u80928.cds, u80928.noncoding, etc.

- Take a look at the type of information in each tab. In particular, take note that:
  - ➢ each of the tabs that contains sequence information contains multiple sequences
  - ➢ the command line you would use to run this program identically to how you just ran it via Jemboss is provided to you under the cmd tab. This will be useful later.

- To work with any of this data further, you have to save it to a local file. Click on the tab with the name ending in **.cds**. Choose the **File | Save to Local File...** option and save this to a location you can find again (e.g. under your bioinf_files directory). Give it a name that will distinguish it from later work -e.g. ***jemboss_bx.cds***. Do ***not*** close the **Saved Results** window as we want to refer to the information under the cmd tab later.

- Go back to the main Jemboss window, go to the **Nucleic | Repeats** section and choose **palindrome** from the list of programs.

- Browse for the file you just saved using the **Browse files...** button next to the box under **Sequence** Filename near the top of the page. Note that you'll have to set the **Files of Type:** option to **All Files** to find your saved file because it has a **.cds** suffix.

- Check that you're happy with all the required options, and give a filename in the **outfile file name** box. For example, *jemboss_palin.txt*. Then press the GO button.

- **Scan through the results to see what has been returned to you.**

You can also view listings of the files on your system using the Jemboss ***file manager*** functionality. Click on the symbol at the bottom right side of the Jemboss window. If you double click on the name of a file that contains text, it will pop up in another window for you to view or edit. Note: the file listings in the Jemboss window are not updated unless you refresh them manually - the regular file browser or the **ls** command are a better way to keep track of what files have been created or deleted.

Using the EMBOSS command line

All EMBOSS commands follow a similar pattern:

- If you just type the command name, then you are prompted for required information.
- If you type the command name followed by **-opt** then you are prompted for optional information as well as required information.
- If you type the command name, followed by a minimum amount of information, and **-auto**, the program runs and uses defaults for anything you have not specified in the command.
- The full command (i.e. the command and all relevant options and values) can be specified by including parameters and arguments on the command line.
- The command name followed by **-h** or **-help** brings up information about the main options for the program.
- The command name followed by **-h -v** brings up information about all options for the program
- Typing **tfm** followed by the command name brings up the full documentation for the program.

So, using the EMBOSS program **seqret** as an example, we could run:

seqret	Run seqret and prompt for required information.
seqret -opt	Run seqret and prompt for required and optional information.
seqret -sequence embl:X03487	Run seqret, specifying the sequence. Prompts for additional information.
seqret -sequence embl:XO3487 -auto	Run seqret, specifying the sequence. Defaults are used for all other options.
seqret -help	Show information about the main options for seqret
seqret -h -v	Show information about all options for seqret
tfm seqret	Show full documentation for seqret

Much more information about the EMBOSS command line syntax is available at:

http://emboss.sourceforge.net/developers/acd/commandline.html

We're now going to run the same tasks we did via Jemboss earlier.

Exercise

- Look at the cmd tab in your jemboss results window for coderet. You should see the following:

```
coderet  -seqall embl:U80928  -outfile jemboss_bx.coderet -auto
```

This command runs coderet, specifies the sequence to use and sets the output file name. The **-auto** option indicates that you do not want to be prompted for further information. This results in default values being used for all options you have not specified on the command line.

- Read about coderet by bringing up the information via the command line:

coderet -h or **coderet -help**	brings up a list of main options
coderet -h -v	brings up a list of all available options
tfm coderet	brings up the full documentation

(EMBOSS commands exercise continued)

● To make things simple, we will edit the command line in the coderet cmd tab of the Saved Results window in Jemboss, and then copy and paste our final command line into a terminal to run the program.

Go to the coderet cmd tab of the Saved Results window in Jemboss, and edit the command to give a new output filename. e.g.

```
coderet  -seqall embl:U80928  -outfile cl_bx.coderet -auto
```

● Open a new terminal window and cd to your bioinf_files directory. Make a new directory to store your result files (as it will make it easier to see what files the program generates by default).

```
mkdir cl_dir
```

● Change directory into your new directory, copy and paste the coderet command line above into the terminal and press the return key. (Recall that we covered highlighting and pasting text using mouse buttons near the end of the first half of this tutorial.) ie:

```
cd cl_dir
coderet  -seqall embl:U80928  -outfile cl_bx.coderet -auto
```

● When the program finishes, list the files in your directory. What has coderet produced? How does this compare with the tabs presented to you when you ran coderet via Jemboss?

You may notice that we have generated a lot of files we don't need. We could have specified to coderet that we only wanted the mRNA sections from the embl entry BX255937. To find out how, you'll need to refer to the coderet documentation (the lists of options won't tell you enough).

● Now run **palindrome** on the mRNA sequence. To do this, you could edit, copy and paste the the command in the Jemboss Saved Results window for palindrome, or you can type palindrome on the command line and answer the prompts. Please run palindrome now, doing one of these.

Once you get to know it, the command line is much faster to get running than programs via Jemboss. However, the power of using the EMBOSS command line is much greater if you need to process groups of files, or do things repetitively.

Below we'll go through an example of running an emboss program on a batch of files using a single command.

If you want to run a job like this repetitively, you can save the commands in a text file and then set things up to get those command executed whenever you want (either by you directly, or by your computer at a time you schedule). We do not cover this in these course notes, but please ask the demonstrator if you would like to know more about this.

Exercise

Fetching a list of sequences using seqret.

- Please look at the contents of the file hexaseqs.list in your bioinf_files directory. e.g. using the command **less**. You will see a list of sequence ids and the database those sequences are in.

- Quit **less**. (hit q)

- We need to tell EMBOSS programs when they are going to work on a list of files rather than just a single file. To do this, we preface the filename with the @ symbol. So, to fetch the list of sequences in the hexaseqs.list file, we can use the command:

```
seqret  -sequence  @hexaseqs.list
```

  The default behaviour of seqret is to fetch sequences in fasta format, with all sequences in a single file with a filename that uses the id of the first sequence. By now you should know how to go about finding out how to alter aspects of the program behaviour like these.

- Take a look at the sequence file you have generated.

You can use this same "list of sequences" syntax with Jemboss. e.g. you could run seqret via Jemboss and specify the sequence name as **@hexaseqs.list**.

General things to keep in mind

If you suspect there may be a more ***efficien***t way to do what you are doing, ***there probably is!***

If you find yourself doing anything ***repetitively***, there is probably an ***easier way to do it.***

Please ***read documentation*** and ***seek advice***. It will ***save you a lot of time*** in the end!

Appendix A: Exercise Answers

1. a) **mkdir testdir**   b) **cd testdir**
2. **cd ../bioinf_files** *or* **cd -**
3. **cp myfirstfile.txt test.txt**
4. **cp mysecondfile.txt subdir**
5. **cp *fil* subdir**
6. **cp test*.embl testdir**

   It is a good idea to use the **ls** command first to check that the files matched are all correct files to move.

   ls test*.embl

   would show the files that would be moved by the **cp** command shown above.

7. **rm mythirdfile.txt**
8. **rm bioinf_files/testdir/myfirstfile.txt**
9. **rm –rf bioinf_files/testdir/subdir**

Appendix B – BLAST references and documentation

Web pages

The blastall and blast+ page in your Bio-Linux Bioinformatics Docs provides links to local web pages with information about NCBI BLAST programs. You can also access this remotely at the URL:
http://nebc.nerc.ac.uk/bioinformatics/docs/blastall.html
http://nebc.nerc.ac.uk/bioinformatics/docs/blast+.html

NCBI BLAST Manual pages
http://www.ncbi.nlm.nih.gov/books/NBK1763/
http://www.ncbi.nlm.nih.gov/blast/blast_help.shtml

NCBI BLAST Web Interface paper
http://nar.oxfordjournals.org/cgi/content/full/36/suppl_2/W5

Sequence similarity statistics
http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html

NEBC BLAST Frequently asked questions
http://nebc.nerc.ac.uk/tools/bioinformatics-docs/other-bioinf/blastfaq

NEBC November 2007 Masters Bioinformatics Course (covers older blastall, rather than BLAST+)
http://nebc.nerc.ac.uk/support/training/course-notes/past-notes/nebc-introduction-to-bioinformatics-msc.-biology-2007

References

The book by Ian Korf is a good place to start in learning about what BLAST can do, how it does it and what BLAST output means. It is now out of date however, and should be read in conjunction with the new blast+ documentation. Also note that wu-blast is now AB-blast, which is licensed software from Advanced Biocomputing LLC.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman.
Gapped blast and psi-blast: a new generation of protein database search programs.
Nucleic Acids Res, 25(17):3389–402, 1997.
Lm05110/lm/nlm Journal Article Research Support, U.S. Gov't, P.H.S. Review England.

S. F. Altschul, J. C. Wootton, E. M. Gertz, R. Agarwala, A. Morgulis, A. A. Schaffer, and Y. K. Yu.
Protein database searches using compositionally adjusted substitution matrices.
Febs J, 272(20):5101–9, 2005. Z01 lm000072-10/lm/nlm Journal Article Review England.

C. Camacho, G. Coulouris, V. Avagyan, M.N. Papadopoulos, K. Bealer and T.L. Madden.
Blast+: architecture and applciations. BMC Bioinformatics, 10: 421, 2009

S. R. Eddy. Where did the blosum62 alignment score matrix come from?
Nat Biotechnol, 22(8):1035–6, 2004. Evaluation Studies Journal Article Review United States.

Ian Korf, Mark Yandell, Joseph Bedell, and Stephen Altschul.
BLAST. ["An essential guide to the Basic Local Alignment Search Tool". Includes bibliographical references and index.]
O'Reilly, Sebastopol, Calif. ; Farnham, 2003. GB A3-Y7706 ill. ; 24 cm.

A. A. Schaffer, L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, Y. I. Wolf, E. V. Koonin, and S. F. Altschul.
Improving the accuracy of psi-blast protein database searches with composition-based statistics and other refinements.
Nucleic Acids Res, 29(14):2994–3005, 2001. Journal Article Review England.

Y. K. Yu, E. M. Gertz, R. Agarwala, A. A. Schaffer, and S. F. Altschul.
Retrieval accuracy, statistical significance and compositional similarity in protein sequence database searches. Nucleic Acids Res, 34(20):5966–73, 2006. Evaluation Studies Journal Article Research Support, N.I.H., Intramural England.

Appendix C – Creating local BLAST databases

Obtaining local BLAST databases

To get the most from BLAST, you should search against a relevant database, which may mean using the relevant parts of a larger database. In general, BLAST searching against the whole of nr or the whole of embl is not a particularly good idea. It takes up your time and computer resources, returns BLAST results with less useful statistics and often less meaningful results. For example, if you are studying marine viruses, do you really care about all the mouse sequence in nr or embl?

Web resources often offer different data subsets you can search against. For example, using the NCBI BLAST pages, you can choose from a certain number of database sections, or you can fine tune the sequence set you blast against using Entrez queries:

http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=FAQ#entrez

http://www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=helpentrez&part=EntrezHelp

Using the EBI BLAST services, you can choose from a number of data subsets, as well as having a choice of WU-blast or NCBI blastall.

http://www.ebi.ac.uk/Tools/blast/

To run BLAST locally, you need to index your collection of sequences; it is these indices that BLAST reads when searching. For some databases or database divisions, you can download prepared BLAST indices from sites such as the NCBI. These are convenient, but do restrict you to searching against particular sets of sequences. It is often useful to create a set of sequences chosen for the types of searches you wish to carry out (e.g. organism or tissue specific) and format them into a database you can search using BLAST.

Any set of fasta sequences can be indexed for BLAST searching. Creating useful sets of sequences is beyond the scope of this course, but two resources to consider are SRS (http://srs.ebi.ac.uk) and Entrez (http://www.ncbi.nlm.nih.gov/books/bookres.fcgi/helpentrez/EntrezHelp.pdf).

For NCBI blastall, the formatdb command is run on fasta formatted files to create BLAST indices.
For BLAST+, the program used is called makeblastdb, and this is the you want to use, though BLAST+ will happily search databases made with formatdb.

Some data resources useful for local BLAST

URL	***Database***	***File format***	***Contents***
ftp://ftp.ebi.ac.uk/pub/databases/fastafiles/uniprot/	uniprot	fasta	Uniprot, swissprot and trembl
ftp://ftp.ebi.ac.uk/pub/databases/uniprot/current_release/knowledgebase/taxonomic_divisions/	uniprot	embl	Uniprot divisions
ftp://ftp.ebi.ac.uk/pub/databases/fastafiles/emblrelease/	embl	fasta	Individual embl divisions
ftp://ftp.ebi.ac.uk/pub/databases/embl/release/	embl	embl	Individual embl divisions
ftp://ftp.ncbi.nlm.nih.gov/blast/db/ ftp://ftp.ebi.ac.uk/pub/blast/db/	various	blast	nr, nt, env and a few other BLAST formatted databases or database sections.
ftp://ftp.ncbi.nlm.nih.gov/genbank	genbank	genbank	Individual genbank divisions

One thing to note in the table above is that uniprot divisions are provided in embl format. However, BLAST indices are created from fasta format files. Unfortunately, the EMBOSS program seqret, which you saw earlier, does not handle entire database divisions well. Instead, you can use a simple script to do the conversion. Instructions on this are below.

If you choose to use pre-formatted BLAST databases, make sure you read the notes about them (usually available as a file called something like REAMDE on the FTP site you get the BLAST files from) as they can be slightly different than the database that results from downloading and formatting your own.

Understand your databases

It is important to read the documentation about the databases you choose to work with.
For example, uniprot and nr are not the same. nt is not a non-redundant database; nr is.
Knowing what is in a database you work with is vital in understanding your results.
Nucleic Acids Research publishes a database issue in January of each year.
This is an excellent resource for finding out more about available database resources.
Another useful resource is the information available via the links on the Library page of SRS at the EBI:
http://srs.ebi.ac.uk/srsbin/cgi-bin/wgetz?-page+top

Building BLAST indices from local sequence files

We will use the uniprot swissprot virus division as an example here. As this is distributed in embl format, and we need it in fasta format, we include a format conversion step in the instructions below.

Bio-Linux machines by default have the BLASTDB environmental variable set to a central location. To find out where it is set to on your machine, you can use the command:

```
echo $BLASTDB
```

If you are logged in as an administrative user, then you will be able to download and work in any area on the machine using your sudo privileges. If you are on a multi-user system and are not an administrative user, the default location for BLAST databases may not be writable by you. In this case, you should talk to your system administrator: either to ask them to give you privileges in the central BLAST database folder, or warn them that you are about to use lots of space in your account for BLAST databases.

These instructions assume that you are working from the directory where you will be storing your BLAST database files. This is not normally the case. Usually, if you download BLAST databases into your account, it is easiest to set the BLASTDB environmental variable to the location of these BLAST databases, and then work from a convenient folder where you plan to store your results. You can set the BLASTDB environmental variable for a single session by typing a line of the form below in the terminal you are working in. To set this variable for every session, you can add the line to your ~/.zshrc file.

```
export  BLASTDB=”$HOME/blastdb”
```

- Download the database section of interest. Here we will work with the uniprot swissprot virus division:

```
wget ftp://ftp.ebi.ac.uk/pub/databases/uniprot/current_release/knowledgebase/taxonomic_divisions/uniprot_sprot_viruses.dat.gz
```

- If you don't already have a sequence conversion tool, download the emblToFastaAndPreProcess.pl script from the NEBC site.

```
wget http://nebc.nerc.ac.uk/downloads/scripts/bioinf/emblToFastaAndPreProcess.pl
```

This script converts embl sequence to fasta sequence. Due to issues that sometimes appear because of the formatting of information in the feature table, it does so by removing the feature lines from the entry before conversion. A version of the script that does not pre-edit the feature lines is also available: http://nebc.nerc.ac.uk/downloads/scripts/bioinf/emblToFasta.pl

- Make this script executable.

```
chmod u+x emblToFastaAndPreProcess.pl
```

- This script can handle compressed files, so you can create a fasta formatted copy of the uniprot_sprot_viruses division by running the command:

```
./emblToFastaAndPreProcess.pl  uniprot_sprot_viruses.dat.gz
```

Notice the **./** at the start of the line. You need this if you are running the script from the directory you are in. There are better ways to do this if you plan to keep this script for use again, but they are not covered here.

- When the script is finished, you should find a file called uniprot_sprot_viruses.fasta in your directory. This is the file we build the BLAST database from.

```
makeblastdb -dbtype prot -in  uniprot_sprot_viruses.fasta  -out sprot_virus
```

- You should now have four new files in your directory: sprot_virus.psq, sprot_virus.pin, sprot_virus.phr and formatdb.log. The last of these lets you know how the BLAST formatting went.

The sprot_virus.p* files are your BLAST indices. You search against them by specifying the BLAST database name **sprot_virus**.

Note:

If you were interested in the swissprot virus division, you would probably be interested in the trembl virus division also. You could download and format that division as described above, and then search the swissprot and trembl virus divisions separately, or as a single, virutal database. Alternatively, you could create a single BLAST formatted database from the two fasta files using formatdb:

```
formatdb -i "uniprot_sprot_viruses.fasta uniprot_trembl_viruses.fasta" -n uniprot_viruses -t "combined sprot and trembl virus divisions"
```

What is the best division depends on what you need to accomplish.

Appendix D: Cheat sheet of basic Linux commands

bg	To send a suspended job to the background
cat ***fileName1***	Output a file to the screen (see also **more** and **less**)
cat ***file1 file2 file3 > newfile***	Append three files together and put the result in newfile
cat -nA ***file1***	Output a file to screen, numbering all lines and revealing non-printing characters
cd ***dirName***	Change to directory dirName. Use **cd ..** to go up one dir or just **cd** to go home.
chmod	To change the permissions or protection on a file, to allow everyone to read a file (chmod a+r somefile)
clear	clear the terminal screen
cp ***fileName1 fileName2***	create a copy of the file called fileName1 and call the copy fileName2
cp ***fileName directoryName***	copy the file fileName *into* a directory called directoryName
cp –R ***dirName1 dirName2***	copy a whole directory called dirName1 and its contents into another directory called dirName2.
date	Print the current date and time
df –h	File system information including space usage
diff ***file1 file2***	Summarise differences between two similar text files file1 and file 2. See also the graphical tool, **meld**
echo $NAME	Print the value of an environment variable called $NAME
emacs	A text editor, more powerful than **gedit**, but more complex.
evince	A command for viewing postscript or PDF formatted files
exit	Exit the current terminal
export NAME=value	Set the environment variable $NAME to “value”
fg	Brings a suspended or background job to the foreground
file ***fileName***	Tries to determine what fileName is by looking at the contents
find -name “test*”	Scans for filenames matching a given glob pattern in the current folder and subfolders. This command is tricky to use. To scan the whole system for files, try **locate.**
gedit	The standard text editor
grep	Search for the occurrence of a pattern
groups *or* **id**	Show what groups a user is in.
head ***fileName***	Show just the first few lines of fileName
history	List log of previous commands you have entered
jobs	Lists any suspended or background processes that you have running. See also **ps** and **pgrep**
kill ***pid***	Kill a process that is running where pid is the process id number (see **ps**). Also consider **pgrep** and **pkill**.
last	Info about who has logged onto the machine recently

less	Type a file to the screen one page at a time (press q to quit, spacebar for next page, b to go back a page)
ls	List the files in your directory
ls –l	List the files in your directory but with “longer” information. (Add -h for more readable file sizes)
man *command*	For help about UNIX command “command”
man -k *keyword*	Lists all UNIX commands that mention the word “keyword”
mkdir *dirName*	Make a directory
more *fileName*	Type a file to the screen a page at a time (press q to quit, spacebar for next page).
mv *file1 dirName*	Assuming dirName is an existing directory, move a file called file1 into a directory called dirName
mv *file1 file2*	Rename file1 and call it file2
nano	A basic text editor that runs in the terminal
passwd	Change your password
pgrep *pattern*	Find process names that contain the pattern. See also **ps**
pkill *processname*	Kill a running process using the process name. Be careful with this! See also **ps**, **pgrep** and **kill**
pwd	Print the full path of your current directory
ps –u	List your current processes
ps –aux	List all processes on the machine. See also **top**
rm *fileName*	Delete a file
rm –rf *dirName*	Delete a directory and all its contents
rmdir	Delete an empty directory
screen	Run the screen manager (read the **man** page first!)
stat *fileName*	Show detailed info on fileName, similar to **ls -l**
tail	Show just the last few lines of a file. See also **head.**
tar -xvz -f *fileName.tar.gz*	Unpack a tarball from the file fileName.tar.gz
someCommand* | tee *fileName	Save output of someCommand to fileName and also print to screen. Use instead of >fileName if you want to redirect but still see the output.
top	List the processes running that are using the most CPU
touch *fileName*	Create an empty file (also updates file timestamps)
wc -l *fileName*	Count lines in fileName
which *commandName*	Reveal what will really be run when you give a command
w *or* **who**	List users currently logged on
yes	A very useful command ;-)
Ctrl-c	Stop (interrupt) a process
Ctrl-r	Interactively search in command log. See **history**
Ctrl-z	Suspend a process, see also **jobs**, **fg** and **bg**