A Short Guide to Unix

Catherine Walsh (with some update on plotting software from Tom Millar)

June 2009 (and June 2013)

Introduction

This is a brief introduction to Unix aimed at users more familiar with the Windows operating system. This guide is designed to get you started quickly using a Unix computer.

Current Unix operating systems have a *Graphical User Interface* or *GUI* which looks and operates similar to Windows. The power of Unix, however, lies in the use of a *terminal* in which commands are entered and processes/programs are started and monitored.

Starting a terminal window varies between Unix distributions. As you would on a machine running Windows, enter the main menu from the desktop screen and search for and open an application called *Terminal*. This will open a mainly blank window with a % or > prompt, something similar to:

```
tjm@star.qub.ac.uk%
```

The text preceding the prompt is the *network* or *local* address of the computer into which you are currently logged. You can type at the prompt in the usual manner, that is, the prompt accepts all keyboard input including text, numbers and symbols. You can erase characters using the backspace key and you can move about the characters already typed using the arrow keys.

Basic Commands

The general form of a Unix command is:

```
% command [-options] [arguments]
```

The bracketed fields are optional, however, some commands do require at least one argument and some also change their function depending on the number and/or types of arguments. To execute the command, hit the **RETURN** key. To check the function of any command, use:

```
% man command
```

This opens the Unix manual in the terminal window and will give you a comprehensive description of the command along with each possible option the command can use and the number and types of arguments the command requires. To scroll through the manual pages, use the **RETURN** key and to exit the manual and return to the command prompt, hit the **Q** (standing for 'Quit') key.

For a short (one-line) description of a command, use:

```
% whatis command
```

To search for command using a keyword, use:

```
% apropos keyword
```

For example to search for commands which are related to printing, use:

```
% apropos print
```

To start with, a simple example:

```
% pwd
```

This command stands for *print working directory* and gives you the name and full path of the current directory.

Another simple command used regularly is:

```
% ls
```

This lists the contents of your current working directory in alphanumerical order. Two options are often used with `ls`:

```
% ls -al
```

The `-a` option (standing for *all*) lists all contents including hidden system files and directories. The `-l` option (standing for *long list*) lists files and subdirectories including file permissions, author, file size and date created or last modified.

This command can be used with any directory as the argument e.g.:

```
% ls /home/docs/files/
```

This will list the files and subdirectories in the directory **files** regardless of the current working directory and is an *absolute* directory address i.e. relative to the *root* directory, '`/`'.

```
% ls -l ../docs/files/
```

This is a *relative* directory address i.e. relative to the current directory. Here '`../`' indicates the parent directory of the current working directory. The current working directory has a similar shorthand, '`./`'.

Two further useful commands are the `cat` command and the `more` command.

```
% cat filename
```

```
% more ../docs/files/output.txt
```

The `cat` command prints the contents of a file to the terminal screen. The `more` command does the same except it prints the contents one page at a time which the user can then scroll though by hitting the **RETURN** key (line by line) or **SPACE** key (page by page). This is useful for long files which do not fit into the terminal screen. To exit the `more` command before reaching the end of a file, hit the **Q** key. These commands are useful if you wish to quickly view the contents of a file *without* altering its content.

Files and Directories

To change directory from the current directory, use the `cd` command:

```
% cd directory
```

For example, if you are in the directory **docs** and wish to change to a directory contained within this called **files**, you can use either the relative pathway or the absolute pathway:

```
% cd files
```

```
% cd /home/docs/files/
```

Using the command `cd` on its own without arguments will take you to the default home directory. The home directory also has a shorthand, '`~/`', i.e.:

```
% cd ~/docs/
```

will take you to the directory called **docs** contained within **home**.

To make a new directory, use the `mkdir` command. You can make a new directory in any existing directory including the current working directory:

```
% mkdir newdirectory
```

```
% mkdir /home/docs/files_2
```

To change the name of a file or directory, use the command `mv`. The first argument is the existing file or directory name, the second is the new file or directory name:

```
% mv oldfilename newfilename
```

```
% mv olddirectoryname newdirectoryname
```

Some examples:

```
% mv outfile.txt outfile_1.txt
```

```
% mv /home/docs/files/ /home/docs/files_1/
```

The `mv` command is also used to move files and directories. In this case, the first argument is the file or directory which is to be moved and the second argument is the new location with, or without, a change in the name of the original file or directory:

```
% mv outfile.txt ../files_1/outfile_1.txt
```

```
% mv /home/docs/files /home/
```

The first example moves the file **outfile.txt** to the directory **files_1** with the new name **outfile_1.txt**. The second example moves the directory called **files** into the home directory alongside its previous parent directory **docs**, without a name change.

To delete a file, use the `rm` command:

```
% rm outfile.txt
```

```
% rm ../docs/files/outfile_1.txt
```

To delete a directory, use the `rmdir` command:

```
% rmdir /home/docs/files
```

If the directory to be deleted contains subdirectories then the option `-r` must append the command (standing for *recursively*):

```
% rmdir -r /home/docs/
```

The last command to do with the manipulation of files and directories is the `cp` command which copies files and directories to the specified location with, or without, a name change. As with the `mv` command, there are many uses and permutations of the `cp` command:

```
% cp filename newfilename
```

```
% cp directory ../newdirectoryname
```

```
% cp filename directory/newfilename
```

Some examples:

```
% cp outfile.txt ./outfile_1.txt
```

```
% cp /home/docs/files/ /home/old_docs/
```

The first example creates a copy in the current directory with a new file name. The second example copies the directory **files** to a different existing directory called **old_docs** without a change in name. In the case of files, if the file specified in the second argument already exists, it will be overwritten, thus care must always be taken when using the `mv` and `cp` commands.

To avoid errors of this nature the option `-i` can be used with both commands to prompt for confirmation that the overwriting can take place:

```
% mv -i outfile.txt ../docs/outfile_1.txt
```

Running Applications

Applications can also be run from the command prompt. The applications discussed are those which will be used in the computer class. In a similar manner to how commands operate, application names are simply typed at the command prompt along with the filename of the file you would like the application to open with:

```
% applicationname filename
```

Most applications in Unix should be set up to run in any directory, however, you may find the absolute or relative path to the application executable is required. To close a running application, **CTRL** and **C** are hit simultaneously. To suspend an application, **CTRL** and **Z** are used. To resume a suspended application, the command `fg` (standing for *foreground*) is used. If you would like the application to run in the background of the terminal, **&** is typed after the filename:

```
% applicationname filename &
```

This leaves the terminal prompt free for further commands. Any application or program running through the terminal is known as a *process*. To list all processes running (useful if you have many applications running in the background), the command `ps` is used:

```
% ps
```

This will list each process running by PID (standing for *Process IDentification*). To terminate a process running in the background, the `kill` command is used in conjunction with the respective PID:

```
% kill 256
```

Once killed, a process cannot be resumed. In Unix, the user has complete control over all processes which are happening!

Text Editors

There are many text editors available in Unix with preferences dependent solely on the user. Examples of text editors are Emacs, Vi, NEdit, KWrite, Nano and Textwrangler (MAC). For quick editing of short files I (Catherine) personally use Nano. Nano is an in-terminal text editor which uses the keyboard to navigate through files and perform functions (i.e. the mouse is not used). To open a file with Nano, at the command prompt type:

```
% nano filename
```

In Nano, changes are saved using **CTRL** and **O** and the file exited using **CTRL** and **X**.

In Unix, there are no restrictions on how files are named (although restricted symbols, such as @ should be avoided). Files which are to be opened with specific applications do not need specific file extensions (unlike Windows). An exception to this are files which will be compiled as computer programs (see later). However, it is good practice (and much clearer for the user) to use conventional file extensions, for example, **'.txt'** for text files and **'.pdf'** for PDF files. It is also good practice to name files and directories without spaces due to the syntax of entering commands on the command line. For example **output_ file_1.txt** is a better filename than **output file 1.txt** as on the command line, the latter would need to be typed as:

```
% emacs output\ file\ 1.txt
```

Plotting

Again, there are many different plotting packages which are available in Unix with the package used dependent on user preferences, the type of data to be plotted and the type of plot required.

The plotting software used in the computer sessions is a well-known, easy-to-use open source program called Xmgrace. Xmgrace allows for a sophisticated approach to data manipulation but here we shall just consider a simple application.

This program allows the user to read in an array (or several arrays) of data for plotting. The array of data to be plotted is a subset of abundances selected from the output file written by the chemical kinetic code. The species to be plotted and their associated abundances as a function of time are generated by a perl script, dc.pl – for fuller details see 'Introduction to Astrochemical Modelling'.

For our purposes, assume that the data to be plotted is contained in the file dc.dat. We invoke Xmgrace from the same directory as dc.dat:

```
% xmgrace &
```

This opens a new window in which all the operations needed to display the data can be carried out through a series of drop-down menus. To read in the data:

```
 Data--Import--ASCII
```

In the new window highlight dc.dat. Load as NXY (the standard method for reading multiple columns data where the x-values - here time in years - are contained in the first column and y-values in the subsequent columns) and click OK.

The plotting window will now show a linear-linear plot of fractional abundances versus time. The axes need to be re-scaled to log-log format to be useful. To do this, go to the main drop-down menu and choose:

```
 Plot--Axes Properties
```

In the resulting menu window, perform the following operations:

Edit X-axis; Scale Logarithmic; Start 1000; Stop 1e8

Axis Label, Label String, Enter: Log Time (yr)

Click Apply at the bottom of the window and now repeat for the Y-axis. In this case choose Start and Stop values of 1e-12 and 1e-3, respectively, and label the axis with the string `Log Fractional Abundance /H2`

The plot will now show the abundance plots of six species, each colour coded, as a function of time. The next step is to customise the appearance of the curves – we can change line thickness, colour, style - solid, dashed, dot-dashed, etc. In the main menu choose:

```
 Plot--Set Appearance
```

The new window that opens contains information each of the 6 abundance curves, labelled (S0, ... S5), in the same order as determined by dc.pl. In our default example, the order is O, OH, H_2O, CO, HCO^+, e^-. When the window opens the first entry (S0) is highlighted. Now we can alter the line properties of this species (O). Default properties are indicated in the box (Type, Style, Width, Color) and can be changed by clicking on the appropriate radio button. Change the default width to 2.0 and associate the proper species name with S0 by entering 'O' in the Legend box. Click Apply and you will see that the black line width has changed and that a small box has opened which indicates that the black line is atomic oxygen. Now repeat for each species in turn, clicking Apply each time and when all six have been labelled click Accept to edit this menu.

Finally, it is useful to label the graph itself with a title, and perhaps a subtitle. In the main menu select:

```
Plot--Graph Appearance
```

Enter a title and subtitle into the appropriate boxes and click Accept. You now have a fully labelled graph on your desktop.

The final task is to save the plot to the hard disk and print it, either directly from Xmgrace or once Xmgrace is closed. In the main menu:

```
File--Selection
```

Enter the name under which the project will be saved, each of these files should be named filename.agr. This file contains all the specific choices you have made in producing the plot and can be opened at a future time for further manipulation, if required.

In Selection, type dc.agr and click OK.

We now generate a file to plot. In the main menu:

```
File--Print Setup
```

You can change Page Orientation and Size if you wish. Click Apply and Accept. In the main menu, select:

```
File--Print
```

to plot the graph on the default printer. The postscript file that is printed is not saved as yet. To save the file in your working directory, select:

```
File--Print Setup--Print to file
```

This saves the file as default.ps where default can be seen in the File name box. Click on this box to change the name of the file to something more memorable. Click Apply and Accept to write filename.ps to your working directory.

Graphics Viewers

The plots generated using Xmgrace will be *postscript* files with the extension '**.ps**'. Postscript graphics can be viewed with a program called GhostView which is invoked using the command `gv`:

```
% gv graph.ps
```

This opens a new window displaying the plot.

Programming

Computer programming is incredibly straightforward in Unix. In basic terms, the program code is created using a text editor and the file given the required file extension corresponding to the programming language used. Examples are **'.c'** for programs written using the C programming language, **'.f'** for programs written in FORTRAN and **'.java'** for programs written using Java.

The program source code is then compiled using the corresponding compiler. This will generate an *executable* file composed of binary code. Running the program involves invoking this executable on the command line and (in theory) the program will then set off and do what it was programmed to! Usually, getting a program to run correctly involves much testing and *debugging* of the source code.

The programming languages used in the computer class are FORTRAN and Perl. FORTRAN (derived from *FORmulae TRANslation*)is a high-level programming language developed specifically for scientific and engineering applications. FORTRAN is used extensively for numerically intensive programming. Perl is another high-level language which provides powerful text processing facilities and as such is often used for the easy manipulation of text files.

In the computer class, the computation is done using a program written in FORTRAN and the results extracted and displayed using scripts written in Perl. It will not be necessary for you to create or directly change any computer code written in either language, therefore, the details of both programming languages will not be covered here. Instead, I will simply describe how a program or script is compiled and executed.

FORTRAN

FORTRAN code is appended with the file extension **'.f'**. The code used is written in a specific FORTRAN language known as FORTRAN 77. There are several compilers which will compile FORTRAN 77 and in the class the compiler used is **gfortran** although the code runs equally well on other compilers such as **g77**:

```
% gfortran program.f
```

This will generate the default executable **a.out** providing there are no compilation errors. To run the executable type:

```
% ./a.out
```

The program will run as required, again, providing there are no bugs in the program code. Recall that `'./'` is a short cut for the current working directory. In the case of running locally generated executables, the location of the executable must be specified.

The default name of the executable can be altered using the `-o` option followed by the replacement name:

```
% gfortran -o program.o program.f
```

Note the name of the executable directly follows the `-o` option. In this case to run the program now type:

```
% ./program.o
```

For complicated programs which consist of many parts or *modules*, a *makefile* is used. A makefile contains the command list for compiling each file or module in turn and links all executables generated into one 'master' executable. To invoke a makefile simply type the command:

```
% make
```

If the makefile is called something other than **makefile**, the name of the makefile must follow the `make` command:

```
% make mymakefile
```

Fortunately, in the computer class a makefile is used to recompile any altered code!

Perl

Perl has powerful text processing and manipulation facilities. In the computer class scripts written in Perl are used to read, extract and process output from the numerical Fortran code. A file written in Perl has the file extension **'.pl'**.

A script written using Perl is simply invoked by typing the name of the script in the command line:

```
% ./perl_script.pl
```

Again, the `'./'` is required to tell the terminal to look for this executable in the current directory.

Unlike FORTRAN code, the Perl code does not need to be compiled prior to execution as the program code contains the following line:

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

This tells the terminal where the required Perl compiler can be found and so it can compile and execute the program in one instruction!

Often Perl scripts accept input from the command line depending on their function:

```
% ./display_graph.pl output.txt graph.eps
```

This script could, for example, take the file **output.txt** and generate a plot called **graph.eps**. Perl scripts are incredibly versatile!

In the computer class, the Perl script dc.pl is used in this way to generate data files for plotting from numerical output. The folder also contains a second Perl script, rate13dc.pl, which generates the ODE subroutine chemical kinetics package from the underlying UMIST Database for Astrochemistry (Rate13), via the command

```
% ./rate13dc.pl -c=H2 -o dcodes.f rate13.rates
```

where -c=speciesname tells the code that speciesname, in this case H2, will have its abundance calculated from a conservation rather than a differential equation. By default, the script also uses charge conservation to calculate the electron abundance. The -o flag sets the name of the output file, in this case a FORTRAN file called dcodes.f and rate13.rates is the name of the file containing the reactions and their associated rate coefficients. It is unlikely that we will have time to cover use of the 'equation writer' in the sessions.

The details of Perl do not need to be covered as all input for the scripts are accepted from the command line as in the example above. To find out the input parameters (if any) a script requires, run the script with zero input parameters. This will display a message explaining the types of input and the order in which they are required for the script to run.

Summary

This table summarises the Unix terms, commands and programs covered in this short guide. The square brackets indicate command options or arguments which are optional.

Command	Description	Syntax
`/`	Root directory	
`./`	Current directory	
`../`	Parent directory of current directory	
`~/`	Default home directory	
`man`	Unix manual	`% man command`
`whatis`	Short description of command	`% whatis command`
`apropos`	Search manual using keyword	`% apropos keyword`
`pwd`	Print working directory	
`ls [-al]`	List contents of directory	`% ls [directory]`
`cat`	Print contents of file to terminal	`% cat file`
`more`	Print scrollable contents of file to terminal	`% more file`
`cd`	Change directory	`% cd [directory]`
`mkdir`	Make new directory	`% mkdir newdirectory`
`mv [-i]`	Move, or change name of, files and directories	`% mv oldfile newfile`
`rm`	Delete files	`% rm file`
`rmdir`	Delete directories	`% rmdir directory`
`cp [-i]`	Copy files and directories	`% cp file directory/newfile`
`ps`	List all current processes by PID	
`kill`	Kill a process running in the background	`% kill PID`
`make`	Compile makefile	`% make [mymakefile]`
`nano`	Opens Nano (in terminal) text editor	`% nano [file]`
`nedit`	Opens NEdit (external window) text editor	`% nedit [file]`
`emacs`	Opens Emacs (external window) text editor	`% emacs [file]`
`gv`	Opens Ghostview graphics viewer	`% gv graph.eps`
`g77`	FORTRAN 77 compiler	`% g77 program.f`
`gfortran`	FORTRAN 77 compiler	`% gfortran program.f`
`xmgrace`	Opens and runs Xmgrace plotting software	`% xmgrace`