A UNIX PRIMER

David A. Clarke

Department of Astronomy and Physics

Saint Mary's University

Halifax, NS, Canada

September, 2002

Revised December, 2005

TABLE OF CONTENTS

I Overview 1

II History 4

III The Basics 5

3.1 Your account 5
3.2 Script files 5
3.3 Directory trees 6
3.4 Listing files (the `ls` command) and the Unix prompt 7
3.5 The "Man" pages 8

IV Common Unix Commands 9

4.1 `passwd` 9
4.2 `mkdir <directory name>` 9
4.3 `rmdir <directory name>` 9
4.4 `ln -s <disc> <link>` 9
4.5 `rm <file or link>` 9
4.6 Creating text files 10
4.7 File names 10
4.8 `pwd` 10
4.9 `cd <directory name>` 10
4.10 `mv <origin file> <target file>` 11
4.11 `cp <origin file> <target file>` 11
4.12 `lp -d <printer> <file name>` 11
4.13 `enscript -d <printer> <file name>` 12
4.14 `chmod` 12
4.15 `gnuplot` 13
4.16 `logout` 14

V Unix Practise Session 15

5.1 Accessing the A&P Unix environment 15
5.2 The *Common Desktop Environment* (CDE) 15
5.3 Practise session on Unix, including GNUPLOT 16
5.4 Logging off, and a warning... 20

A UNIX PRIMER

I OVERVIEW

As an instructor teaching second and third-year physics students, teaching *computing programming* poses a unique challenge. While some of you may already be sophisticated computer programmers (and quite possibly much more sophisticated than your instructor!), others of you may be blown away by the idea that the keyboard can be used for something other than entering your ID and password, and composing text for e-mail and assignments. And yet you all come to me at the same time looking for enlightenment!

My challenge, then, is to get those of you who may fall into the second category up to speed fast enough to get you programming usefully without boring to death those of you in the first category, and to this end I have created this primer. For the computing sophisticates amongst you, this document may serve as nothing more than a handy reminder for the one or two things you may have forgotten. To the rest of you, this is what you need to read, digest, and learn in order to catch up enough to your colleagues to compete with them in undergraduate computational physics. Believe it or not, this can be done in a matter of weeks, not months; you just need to relax, open your mind, and realise the world of computing is much more than just "point and click"!

One day when my son was 13, he popped into my home office while I was working and saw me typing at my keyboard. "E-mailing again Dad?" he asked. "No, programming", I responded. He looked perplexed. "You know, a set of instructions to make the computer do what I want." It was immediately apparent that my quick answer was of zero help. "So why don't you just use the mouse?" he asked, truly believing, I have to guess, that his suggestion would come as a revelation to me.

So, it is to those of you who were raised to believe that the mouse is the only real way to communicate with the computer that this primer is directed. This primer won't get you to the point of where you can actually generate your own scientific program; for that you will have to read my FORTRAN Primer after you have become familiar with this one. Instead, this Primer is designed to introduce you to the computer environment known as the "Operating System", a so-called mid-level language in which you can type (using the keyboard instead of the mouse!) commands of complexity, detail and usefulness that will open up the computer in a way that a three-button mouse simply cannot.

So, to get us started, I ask some questions. Have you ever wondered how the computer actually puts that little icon that you click on in the spot that it appears in? How did it come to look as it does; its design, it's colour, where it appears on the screen? Why does clicking on it "open it up", and what does "opening up" actually mean? Yes, the opened "application" occupies more of your screen, but how does the computer know how big to make it? How does it know that a certain field accepts text while another doesn't, and

how does it know what to do with that text? If your answer is "By using some web design software.", then I ask you right back: How was the first web design software created?

Of course, all these things are created by people. Computers don't have these features by virtue of being formed in the factory from silicon and plastic. At some point, the programmer has to communicate with the computer at a level in which he or she can actually tell the computer what to write where and how, where in memory certain data reside, and how to translate a computer signal into an action by another machine (*e.g.*, a printer, a telescope motor, a robotic arm, a fuel injector, even your computer screen). To do all this, a user needs to access what is called the "Operating System", or OS for short. The OS is itself a computer program that someone had to write, and OS programmers create OSs at an even more fundamental computer level called "assembly language". This is beyond the scope of this Primer, and indeed, most scientists engaged in scientific programming never have to learn the first thing about machine-level (assembly) type languages.

To a Computer Systems Administrator, an OS is something they have to "install" on a computer before the computer can even boot up. Ultimately, it is what tells the computer what the keys on the keyboard mean, how to deal with incoming and outgoing signals, how to interpret the streams of 1s and 0s as information, how to access discs, and so it goes. To the user, an OS is a system of allowed commands that may be typed in at the keyboard to accomplish rudimentary tasks. While this is a small part of the OS, it represents 99% of what you and I see as the OS, because that is the part we have to learn to use the computer; everything else is (thankfully) transparent. So it is the small portion of the OS that allows us to issue commands that this Primer introduces.

The commands of an OS essentially do two things. They create files and directories on discs (literally, by assigning a portion of the available disc space to your purpose), and they destroy files and directories (a disc is to a directory is to a file what a filing cabinet is to a drawer is to a file folder). Moving a file from one directory to another is like a create (elsewhere), then destroy (here). Copying a file is like a move without the destroy step. When you create a file, you create more than just a place to store data or text (which could be a list of numbers, a letter home to Ma, or a computer program to solve the equations necessary to simulate the first three minutes after the Big Bang, if you knew what these equations were that is!). When you create a file, you also create a place to store its name, its size, when it was created, *etc.* These "file extensions" are all created automatically. Thus, renaming a file is simply destroying the name file extension, then creating a new one. Asking the OS to tell you what files are in a directory is asking the OS to copy (*i.e.*, create elsewhere) all name file extensions into another file, and copy the contents of this file not to the disc, but to your computer screen.

Thus, the list of OS commands and structures appear complicated only because their developers have become really good at making imaginative uses of creates and destroys. So if you start to feel bogged down in the details of the OS, it may help to remember that all commands in an OS boil down to a create and/or a destroy (a 1 or a 0).

For scientific computing, the most popular OS in the world by far is Unix. Those who may be familiar with the rudiments of DOS may recognise some of the Unix commands,

since there is some overlap between DOS and Unix. Unix is a multi-user, multi-purpose computer operating system that allows more than one user to use a machine at the same time more or less transparently. Each user would be logged in at their own "terminal" (which could be nothing more than a keyboard, a monitor, and an internet line, another full-fledged computer, or the keyboard and monitor of the actual machine) and may not be aware of other users at all.

In order to accomplish this, Unix was designed with the ability to "swap" applications (*e.g.*, your computer program, Netscape, a clock, the window in which you are typing, *etc.*) in and out of "memory" and on and off the processor (Note that these are just "moves", which are creates and destroys). One pleasant side-effect of this ability is "virtual memory", in which disc space can be used as "memory" if there isn't enough "real" memory installed on the machine for your application. This may be slower than if there were enough real memory, but better late than never!

II HISTORY

Unix was developed in the late 1960's by Ken Thompson and his collaborators at the Bell Laboratories (now AT&T) in the U.S. Unix is not an acronym, but a name that was chosen rather whimsically and ended up sticking. At the time, the prevailing OS at Bell Labs was "Multics" which emphasised the *mult*iple usages of the new electron*ics*. Multics was extremely cumbersome, and understood by few. A newer, simpler system was developed and called "Unics", where the *Un* as in *Un*ity was to connote a simpler system than *Mult* as in *Mult*iple and the fact that the new name was a homonym for "Eunuchs" (and thus the emasculated version of Multics) was an intended pun. Sometime later, the more compact spelling "Unix" was adopted.

To no one's greater surprise than the developers, Unix ended up being adopted by numerous groups, and has since become a standard multiple-user OS throughout the world. At first used primarily by the scientific community, Unix is now widely used in business and is finding its way into the home, with many home computers now able to support versions of Unix such as "Linux". Our own system in the Department supports both Linux and the flavour of Unix developed by SUN Microsystems: SOLARIS. As both Linux and SOLARIS are Unix compatible, Linux on the PC will "look" identical to SOLARIS on the Suns (*i.e.*, command syntax and what they output are the same), so there is no need to learn two OSs.

III THE BASICS

This purpose of this document is to introduce the reader to the very basic fundamentals of Unix. Having said that, this document also contains more than 90% of what I know about Unix, so it is quite possible to function within Unix without having to absorb all there is to know.

For those who thirst for more, there is no shortage of resources. The Web is a semi-infinite source of current information, but it changes daily and there is no point in my trying to list various Unix websites here; they would be dated and possibly off-line as soon as this document gets printed. Instead, I suggest you simply get on your favourite web browser, and search for 'Unix', 'Unix tutorials', 'Unix for dummies', *etc.*; you'll find thousands of hits. There are also numerous books in print that come and go with the seasons. Any decent bookstore will have *something* on Unix to get you beyond this primer.

3.1 Your account

Unix manages the "accounts" of many users on the same machine. Your "account" consists of a space on the "hard disc" to which the OS maintains a "pointer" linking your username and password (how Unix "knows" you) to your account. In your account, you may store various files which you may or may not choose to arrange in a "directory tree" of your construction, and you may also issue commands. These commands may be known commands to the machine, or commands you create by generating "script files", by downloading software from the internet, or by creating, compiling and linking your own software written in one of many computer languages (*e.g.*, FORTRAN, C++).

Your account already comes with several files, many of which are "hidden" files, whose names are preceded with a period. DO NOT REMOVE THESE!!! The `.login` (read dot-login) and `.cshrc` (read dot-c-s-h-r-c) files control how your account "behaves"; that is, it sets "aliases" (so that you can type in something you'll remember for a Unix command that is too long to remember), and controls defaults for your mail, your prompt, where output goes, what permissions are given to new files, *etc.*

3.2 Script files

The `.cshrc` and `.login` files are special examples of what are called "script files". Script files are those which contain in succession a series of Unix commands that you could very well type in at your screen each time you want its functionality. Thus, each line in the `.cshrc` file represents a valid, stand-alone Unix command. Script files are useful, then, to submit the same set of (possibly lengthy) commands each time you open a new window, or submit a job, or prepare a plot. Files with a period at the beginning of their filename are normally files that the system recognises as having to be executed at certain key times. Thus, `.login` is a script file executed every time you log onto the system, `.cshrc` is executed each time a new window is opened, `.mailrc` is executed every time your mail tool is invoked, *etc.*

At first, you may scarcely be aware of the `.login` and `.cshrc` files, and even less aware of the need to create your own script files. However, as you gain experience in Unix, you may start asking questions like: 'Is there any way to rename a series of similarly named files all at once, rather than having to rename each one individually?', or 'Is there any way to log onto a remote system, retrieve a given file from the remote storage device, compress it, transfer it to my local site, log off the remote site, decompress the file locally, and then prepare the file to be examined with software running on my local machine?' To both of these questions, the answer is the same: 'Yes, with script files.', and as the need for these files becomes apparent, you will be motivated to find out how to create them.

For those who may wish to start creating script files now, it is sufficient to know that they are simply text files which contain the series of normal Unix commands, however complex, that you would otherwise type in at your computer screen. These files are then placed on your disc, given execution permission (*e.g.*, `chmod 755 <scriptfile>`, see the `chmod` command below), and executed by typing in the name you gave the script file at the cursor, followed by the `Return` or `Enter` key. Script files you create should not be given names starting with a period. Let these be reserved for system script files such as the `.cshrc` file. Instead, the names of your script files should start with an alphanumeric character. The length of the script file name can be as long as you like, though it is practical to keep the names to fifteen characters or less. Script file names may *contain* the punctuation characters `-`, _, or `.`, but they shouldn't be *started* with one of these characters.

3.3 Directory trees

As you use your account more and more, you will be creating more and more files, and after you've accumulated 30 files or more, you should think about making a directory tree. My "home directory" (that which I arrive at automatically when I log on) is called `/disc6/home/dclarke`, or equivalently, `~dclarke`. Yours will have a different disc number and of course your name instead of mine. Your home directory is the area that you "own", that is where you can create and destroy files at will. As discussed below, you may use the `mkdir` and `rmdir` commands to create and remove directories as needed, and then the `mv` (move) command to move files from one directory to another.

Thus, when naming a directory (and its sub-directories), bear in mind the common theme each file will have in that directory path, and then name that directory accordingly. Thus, in your home directory, you may have the directories `homework`, `e-mail` (hyphens are allowed punctuation inside the name, not at the beginning), `jpegs`, `personal`, *etc.* Inside the directory `homework`, you might have created sub-directories `PHYS2335`, `PHYS3437`, *etc.* You might leave read-permission open to all other users on all your directories (useful if you were working with others, or if you want me to have access to your files), with the possible exception of the directory `personal`, to which you may wish to restrict read permission to yourself (*e.g.*, `chmod 700 personal`; see page 11). A word of warning: the System Administrator can read all files—permission protected or not—and we do snoop (to be sure the system is not being abused), so don't keep things that are too personal here!

3.4 Listing files (the `ls` command) and the Unix prompt

The Unix command for listing the contents of a directory is `ls`. All Unix commands, including `ls`, have a list of options (long list for some commands, short for others) that must be specified with the command. Fortunately, most options have sensible defaults, and often one can just type the command without any options. For example, when I type `ls` on my machine inside my directory `~dclarke/stmarys/courses`, this is what my terminal session looks like:

```
titarus 44> cd ~dclarke/stmarys/courses
titarus 45> ls
ASTR5613       PHYS1210       PHYS2335       PHYS3436       PHYS4472
ASTR5695-6696  PHYS1211       PHYS3435       PHYS3437       PHYS4500
titarus 46>
```

The "`titarus 44>`" part is how my machine prompts me. I have it set up so that the machine name and the number of commands issued since my current login session began are strung together with the `>` symbol to form a "prompt" which appears at the far left of each line. To do this, I placed the lines:

```
if ($?prompt) then
  set     prompt = "`hostname` \!> "
  set     lineedit
  echo    ""
endif
```

at the end of my `.cshrc` file. The cursor (a small triangle, a flashing underscore, or whatever your computer does to indicate where the next character typed from the keyboard will appear on the screen) appears just to the right of the prompt, and commands are issued from there.

After typing `ls`, I hit the `ENTER` key, and the above output is the result. In this case, I have listed all files in this directory, each of which happen to be a directory containing files pertinent to the course whose name is used to name each directory. Note that there is no way of knowing whether an item is a file or a directory by looking at its name, at least when you use `ls` with no options.

If instead I type `ls -F`, this is what I get:

```
titarus 46> ls -F
ASTR5613/       PHYS1210/       PHYS2335/       PHYS3436/       PHYS4472/
ASTR5695-6696/  PHYS1211/       PHYS3435/       PHYS3437/       PHYS4500/
titarus 47>
```

Options are specified on the command line, and prefaced with a minus sign (`-`). In this case, `-F` causes a slash (`/`) to be appended to the end of directory names and an asterisk (`*`) to be appended at the end of executables (names of compiled and linked programs that you

can run simply by typing in their name followed by the ENTER key). Note that -F would typically mean something different for each command that includes it as an option.

I can then direct the Unix command to act only on one of the directories. For example, to list the contents of the subdirectory PHYS3437:

```
titarus 49> ls -F PHYS3437
fprimer.latex   fprimer.ps      syll06.ps       uprimer.latex
titarus 50>
```

These are all files (no slash appended). The specification of the directory or file, as the case may be, does not take the minus sign, and follows all options which do take the minus sign.

Another useful option for ls is -l:

```
titarus 50> ls -l PHYS3437
total 1228
-rw-r--r--   1 dclarke  staff     116743 Nov  6 13:33 fprimer.latex
-rw-r--r--   1 dclarke  staff     322676 Nov  6 13:34 fprimer.ps
-rw-r--r--   1 dclarke  staff       1689 Jan  7 13:20 syll06.ps
-rw-r--r--   1 dclarke  staff      28208 Jan  7 19:10 uprimer.latex
titarus 51>
```

More information is given about each file. The first columns (-rw-r--r--) indicate permission settings (who may read, write, and execute, and who may not; see chmod on page 11). This is followed by the "owner" (in my case, dclarke), the "group" to which the owner belongs (in my case, staff), the size of the file in bytes, the date and time of the last update to the file, and finally the name of the file.

3.5 The "Man" pages

For any command, you can find out much more than you would ever need to know about it by accessing the "manual pages", or "man pages" for short. So, for example, by typing man ls, you will get on-line help for the command ls. A "man page" may actually be many screens of text, and only enough text to fill one screen will appear at a time, with a percentage of the file shown appearing at the bottom. You can scroll up the next screen of text by typing the space bar, and if you want to quit out of the man page without going through all screens, simply type q. You don't even need to hit the ENTER key; the system quits out of the man page as soon as you type q. While the man pages can be long, you soon get used to skipping through them until you find what you want.

IV COMMON UNIX COMMANDS

The commands listed here represent the vast majority of all commands I ever use; yet they represent only the tip of the Unix iceberg. Unix commands are case sensitive with all commands expressed in lower case, and with upper case used only for some command options. Thus the command to list files is `ls`, not `LS`. As discussed in §V, each Unix command may be used as part of a script file, and you will all no doubt be using this feature of Unix before long.

4.1 `passwd`

This command allows you to change your password. The first time you log onto the Unix system, issue this command. You will be asked to `Enter login password`. This is asking you for your **present** password, not your new one. After typing this in correctly, it will prompt you for your new password, then it will ask you to type this in a second time for confirmation. The entire system will then be told of your password change, which may take a few minutes. This won't matter for your current session, but if you were to try to log onto the system somewhere else immediately after changing your password, you may have difficulties until the new password filters throughout the network.

4.2 `mkdir <directory name>`

Creates a new directory with the name specified. The angle brackets are not to be included in the name of the directory; the brackets only indicate you should change the text inside to the name you want. *e.g.*, when preparing for this course, I issued the command:

```
mkdir PHYS3437
```

to make the directory to contain all the files I will need for this course.

4.3 `rmdir <directory name>`

Removes an empty directory.

4.4 `ln -s <disc> <link>`

This command creates a so-called logical link to a directory on some other disc (here or in Tuktoyuktuk) and creates the link `link` in your directory that can be treated as a subdirectory. Thus,

```
ln -s /disc45/dclarke temp
```

will create a link called `temp` that will appear as a subdirectory when all files are listed (`ls -F` will list it as `temp@` with the `@` symbol identifying `temp` as a link). Thus, typing `cd temp` will switch you to the named directory on the other disc just as simply as `cd <directory>`

will switch you to a different directory on the current disc. This is very useful since your home disc space (that which is backed up regularly) will be rather limited, and for jobs requiring larger disc storage, you will be given access to "temporary disc space" which is not backed up, is purged once in a long while (with ample warning), but is semi-infinite in scope. This will be on a physically different storage device and to access it quickly, you can establish a link between it and your home directory using the `ln -s` command. Note that `ln` requires the `-s` option in order to establish the link properly.

4.5 `rm <file or link>`

Removes a file or a link. The system will ask you if you are certain you want the file or link removed before removing it. To suppress the "Are you sure?" prompt, add the option `-f`. Personally, I prefer to use `rm -f` all the time, and since it is a pain always to have to remember the `-f`, I have added the line

```
alias rm "rm -f"
```

to my `.cshrc` file so that when I type `rm`, I actually get `rm -f`.

Note that `*` is the file name wildcard. Thus, unless you are very *very* **very** sure of yourself, resist the temptation to issue the command: `rm -f *` !

4.6 Creating text files

To create a text file, you invoke your favourite text editor, such as `vi`, `emacs`, or `te`, type what you want, tell the editor what name to give the file, then exit the editor. Your file will then reside in the directory in which the editor was invoked. There are man pages for `emacs` and `vi`, but none for `te` (which is a SOLARIS Openwindows editor). If you don't know any of these already, I would recommend learning `emacs`. `emacs` comes with *all* Unix systems, and could end up being the last text editor you ever have to learn. It is mercifully self-explanatory, though there are some key control characters you have to learn.

4.7 File names

The names of files and directories follow the same rules. The length of the name you assign is virtually unlimited (but brevity is always a good thing!). Any upper or lower case letter may be used, along with any digit and the punctuation `. - _` (period, hyphen, underscore), with the proviso that file names and directory names should not start with the punctuation. No other punctuation should be used. Note that Unix is not totalitarian enough to prevent you from using other punctuation, or from starting a file name with a hyphen. Therefore, if you do start a file name with a hyphen, you might wonder how Unix can then tell the difference between your filename, and a list of options you might be trying to give it. Answer: it can't, and things can get awfully confused when it tries!

4.8 `pwd`

"Present working directory", indicates in which directory you are currently working.

4.9 `cd <directory name>`

Changes directory to the directory named. If the named directory does not start with a slash (`/`), it is assumed that the first directory in the string given is a subdirectory of the "present working directory". If the first character is the slash, then the directory name is assumed to be an "absolute path". Finally, the double dot (`..`) indicates "one directory up". Examples:

`cd PHYS3437`
: changes directory to `PHYS3437`, and is successful so long as `PHYS3437` is a subdirectory of the present working directory.

`cd /disc6/home/dclarke/stmarys/courses/PHYS3437`
: is the same thing, but uses the absolute path name (as given by `pwd`, for example). This command would be successful regardless of the directory from which it was issued. On most systems, the "tilde" (`~`) can be used in place of the "`/disc6/home/`" bit. Thus,

  `cd ~dclarke/stmarys/courses/PHYS3437`

  is an equivalent command.

`cd ..`
: Moves up one directory.

`cd ../..`
: Moves up two directories.

`cd`
: Gets you to your top home directory. It is a short form for `cd /disc6/home/dclarke`, or equivalently `cd ~dclarke`.

4.10 `mv <origin file> <target file>`

Moves a file from one location to another. `origin file` could be an existing file in one directory that you wish to move to another directory. *e.g.*,

`mv PHYS3437/assign.01 PHYS2335/assign.01`

This command moves the file `assign.01` from directory `PHYS3437` to directory `PHYS3435` keeping the same name, and would be issued from the parent directory to both (*i.e.*, the directory from which, when `ls` is issued, both these directories are listed). So long as the name of the file is to be left unchanged, the file name `assign.01` in `target file` is optional. However, if you wanted to change the name of the file while you were moving it, you could specify the new name in `target file`.

`mv` can also be used to change the name of a file. *e.g.*,

```
mv assign.01 assignment1
```

would change the name of the file containing the first assignment from `assign.01` to `assignment1`. This command as given would have to be issued from inside the subdirectory in which the file `assign.01` resides (`PHYS3437` in this case). If I were in the parent directory of `PHYS3437`, I would type instead:

```
mv PHYS3437/assign.01 PHYS3437/assignment1
```

4.11 `cp <origin file> <target file>`

Copies `origin file` to `target file`. Directory trees may form part of the file specification, just as for `mv`. This command is very much like `mv`, except the original file is left intact.

4.12 `lp -d <printer> <file name>`

sends the postscript file `file name` to the printer `printer`. `printer` can be one of `lj1`, `lj2`, *etc.* Type `resources` at your prompt to get a list of departmental resources, including names and locations of printers. Some printers may be inaccessible to you, so please be sure you know where you are printing before you do so. To get a list of applications (computer packages) available throughout the departmental network, type `apps` at your prompt.

NB: This command is for postscript files only. If you don't know what postscript is, you shouldn't be using this command!

4.13 `enscript -d <printer> <file name>`

sends non-postscript (*e.g.*, plain text files) to the printer. For me, the word `enscript` doesn't exactly jump to mind when I want to print something. So I include in my `.cshrc` the line

```
alias enscript print
```

allowing me to type `print -d <printer> <file name>` instead.

4.14 `chmod`

Files can be protected from other users, if this is your desire. To see the file protection settings of a given file or set of files, type

```
ls -l <file(s)>
```

where `file(s)` could be a directory name, in which case all files contained by that directory would be listed. If the file you were examining was called `uprimer.latex`, then the `ls -l` command would generate the following output:

```
-rw-r--r--   1 dclarke  staff      28208 Jan  7 19:10 uprimer.latex
```

Of interest here are the first ten characters only.

The first character is a hyphen for files (as above), and a '`d`' for directories.

The next three characters (`rw-`) lists the permissions for me, the creator and owner of the file. Thus, I have permission to read (`r`) the file (which includes loading it into a text editor, copying or browsing it, *etc.*) and to write to it (which includes adding text, modifying existing text, or deleting it completely). The third field is a hyphen, meaning I have not given myself execution privilege (not needed for a plain text file, but needed for a directory or an executable file).

The next three characters (`r--`) indicate the permissions for others in my "group", which is defined by the System Administrator. For me, this includes all other faculty members in the Department but not the students or staff. For you, your group will likely include all other undergrads, but not the graduate students, staff, or faculty. As it stands, I have given my group read permission, but not write or execute. For obvious reasons, it is rare that you would ever give anyone else write permission to your files.

Finally, the last three characters (`r--`) indicate permissions for the "world", namely all other users of the system outside yourself and your group. If you ever think you will need me to read a file of yours (highly likely!), make sure that "world" (which includes me) has read permission. NB: for someone in "world" to read a given file, read permission must be given to that file, and read-execute permission must be given to all directories in the directory branch containing that file up to your home directory.

Using the `chmod` command and the following code, you can change the permissions for owner, group, and world for any given file(s) or directories that you have created. Since you did not create your home directory (*e.g.*, `~dclarke`), you cannot change the permissions of your home directory. Thus the command:

```
chmod <ogw> <file(s)>
```

will change the permission of the file(s) or directory named, where `o`, `g`, and `w` are integers between 0 and 7 giving permissions to the owner, group, and world respectively, and where

0 ⇒	no read, no write, no execute	4 ⇒	read, no write, no execute
1 ⇒	no read, no write, execute	5 ⇒	read, no write, execute
2 ⇒	no read, write, no execute	6 ⇒	read, write, no execute
3 ⇒	no read, write, execute	7 ⇒	read, write, execute

Thus,

```
chmod 640 uprimer.latex
```

will give me read and write permission, my group read permission, and world no permissions at all for the file `uprimer.latex`. NB, if you haven't given yourself write permission on a file, it will be impossible to modify or delete it. Some people use this feature to protect very valuable files from being changed or erased by mistake.

When a new file is created, how is the permission decided? The System Administrator chose a default for you when your account was created, and normally this is 644 (755 for executable files). Sometimes, however, 600 (700 for executable files) is the default, which makes it a pain to share files (each time you create a file you have to remember to change the file protection too). To overcome this, you can add to your `.cshrc` file the statement:

```
umask <OGW>
```

where, annoyingly enough, `O` = 7 − `o`, `G` = 7 − `g`, and `W` = 7 − `w`. Since there is no harm in giving ordinary text files "execute permission", one need not worry about the distinction between 755 for executable files and 644 for text files; use the same `umask` for them all. Thus,

```
umask 022
```

will give all new files created permission 755, as desired if you wish to give others in your class and myself read access to all files created since the `umask` command was last invoked (and thus the need to put it in your `.cshrc` file so that the `umask` command is issued every time a new window is opened).

4.15 `gnuplot`

Fires up `gnuplot`, the public domain 2-D and 3-D plotter available on the Astronomy and Physics network. A typical session on gnuplot might look like the following:

```
crux 51> gnuplot
gnuplot> set term post                      needed for making hardcopy
gnuplot> set output "file1.ps"              output will be written to file1.ps
gnuplot> plot [-pi:pi] [-1.1:1.1] sin(x)
gnuplot> set term x11                       needed to resume putting plots to screen
gnuplot> set output                         needed to resume putting plots to screen
gnuplot> splot [-2:2] [-2:2] 3*x+2*y+4
gnuplot> quit
crux 52> lp -d lj2 file1.ps
```

where the last command prints the file you created to `lp2`, the printer in MM312.

Inside `gnuplot`, you can type `help` at the `gnuplot>` prompt, and get help on various keywords. For more information, Henri Gavin's short primer on GNUPLOT is available from `http://www.ap.smu.ca/~dclarke/PHYS3437`.

4.16 `logout`

You *must* log out at the end of your session. Unattended terminals with open accounts put the entire system at risk, and users whose accounts are found left open will have their privileges terminated without warning. `lo` is a valid short form for this command.

V UNIX PRACTISE SESSION

5.1 Accessing the A&P Unix environment

From an Astronomy and Physics PC:

1. `<Control><Alt><Del>` to log onto the PC. Enter your PC userid and password in the appropriate fields and make sure "`DOMAIN`" is set to "`astphy`". This logs you into a `WINDOWS NT`^{TM} environment, with various icons scattered on the left of the screen.
2. Double click on the icon "`X-Win32`", which will clear most of the screen leaving a box with all the available Unix machines listed.
3. Double click on `crux`.
4. Screen is blanked out, and a Unix login box appears. Type in your unix userid and password (possibly, though not necessarily the same as your PC userid and password) in the appropriate fields, then click on `OK`.
5. Screen assumes the "Common Desktop Environment" (CDE) configuration (window-based).

From an Astronomy and Physics Sun Workstation:

1. The Unix login box referred to in step 4 above should be waiting for you. If the screen is black, wiggle the mouse around; the screen-saver may well be on. Type in your unix userid and password in the appropriate fields, then click on `OK`.
2. Screen assumes the "Common Desktop Environment" (CDE) configuration.

From outside the Department, how you access an external Unix box from your screen (be it from one of the University computer labs, home, or anywhere there is internet access) will depend entirely on how your computer is set up, and whether or not your computer and the Departmental system know about each other. You will have to talk to one of the Departmental computer technicians and/or the person who set up the remote computer you are using for details.

The remainder of this section is written under the assumption that you have successfully logged onto the Departmental Unix system (SOLARIS) and that you are using the CDE configuration.

5.2 The *Common Desktop Environment* (CDE)

Once in the CDE, you are ready to begin your Unix session. In the CDE, most of your usage will be within either the "console" or "text editor" windows. If these do not come up

automatically when you enter the CDE, you can raise one by making the appropriate selection from the "toolbar" at the bottom of your screen. Each item in the tool bar refers to a greater selection which you can uncover by clicking with the left mouse button on the arrow. Under "`cpu disk`", you will find "console", and under the icon looking like a pencil and paper, you will find "text editor". With the menu raised, click on the desired application with the left mouse button.

The console window is where you enter Unix commands, such as those listed in §IX of this primer.

The text editor window is where you can create and edit ASCII files, appropriate for writing computer programs, Unix script files, LaTeX files, e-mail, *etc.* ASCII is "ordinary text" (no highlighting or formatting characters common on Microsoft Word™ documents for example), and consists of 256 (2^8) internationally-agreed-to characters including all symbols on the QWERTY and the French AZERTY keyboards, among others.

Finally, the CDE supports four independent "desktops", through which you may toggle by clicking in the centre of the toolbar at the bottom of your screen. This allows you, for example, to have a desktop for your programming, a desk for graphics, a desk for web development, *etc.*, if you are so inclined. Regardless of which desktop you may be using, you access the same files and directory tree; namely those on your account. The states of these desktops are stored when you log off so that you may resume them as you left them the next time you log on.

5.3 Practise session on Unix, including GNUPLOT

The following is a sample Unix session. To start familiarising yourself with the Unix environment, follow these steps on your machine. All computer generated text, messages, prompts, *etc.*, are indicated in `typewriter font`, all user (that's you!) input is underlined, and all comments that are neither computer generated nor intended for you to type in are in *italics*. The first messages are those generated as soon as you have successfully logged on. The exact wording of some of the text (*e.g.*, the prompt) will depend upon how your environment (*e.g.*, `.login` and `.cshrc` files) is set.

```
Executing system.cshrc on olympus ...
Thu Dec 29 11:51:59 AST 2005
/disc6/home/dclarke

You have mail.
olympus 51> passwd                                  First thing: change your password!
passwd:  Changing password for dclarke
Enter login(NIS+) password:                         Type in your current password.
New password:              Use eight characters or more, mix cases, use punctuation.
Re-enter new password:
NIS+ password information changed for dclarke
```

```
NIS+ credential information changed for dclarke
olympus 52> pwd
/disc6/home/dclarke
olympus 53> ls -F
ASTR5613/       disk1@          meetings/       proposals/      talks/
PHYS2335/       documents/      nmlst/          radio/          test.f
a.out*          hdf/            ncar/           referee/        testgnu
bin/            latex/          pals/           sci/            texput.log
consult/        letters/        papers/         setup.dvi       vita/
datavu/         library/        plotz/          setup.log       work/
default/        mail/           preprints/      stmarys/        zeus/
olympus 54> mkdir PHYS3437                              Making a new directory.
olympus 55> dir                        I have alias dir "ls -F" in my .cshrc file.
ASTR5613/       disk1@          nmlst/          referee/        texput.log
PHYS2335/       documents/      ncar/           sci/            vita/
PHYS3437/       hdf/            pals/           setup.dvi       work/
a.out*          latex/          papers/         setup.log       zeus/
bin/            letters/        plotz/          stmarys/
consult/        library/        preprints/      talks/
datavu/         mail/           proposals/      test.f
default/        meetings/       radio/          testgnu
olympus 56> cd PHYS3437
/disc6/home/dclarke/PHYS3437
olympus 57> ls -F                                   No files in the new directory yet.
olympus 58> mkdir psfiles          Create a subdirectory psfiles within PHYS3437.
olympus 59> cd psfiles
olympus 60> cp ~dclarke/stmarys/courses/PHYS3437/psfiles/* .
```

You should actually type: "`cp ~dclarke/PHYS3437/psfiles/* .`*", since my directory* `~dclarke/stmarys` *is read protected. The asterisk means "everything", and the period means "to here".*

```
olympus 61> ls
dprimer.ps      gnuplot.ps      projects.ps     uprimer.ps
fprimer.ps      lprimer.ps      syll06.ps
olympus 62> ls -l
total 576
-rw-r--r--   1 dclarke  staff      59573 Dec 29 13:48 dprimer.ps
-rw-r--r--   1 dclarke  staff     357079 Dec 29 13:48 fprimer.ps
-rw-r--r--   1 dclarke  staff     930107 Dec 29 13:48 gnuplot.ps
-rw-r--r--   1 dclarke  staff     424826 Dec 29 13:48 lprimer.ps
-rw-r--r--   1 dclarke  staff     113184 Dec 29 13:48 projects.ps
-rw-r--r--   1 dclarke  staff      44142 Dec 29 13:48 syll06.ps
-rw-r--r--   1 dclarke  staff     172907 Dec 29 13:48 uprimer.ps
olympus 63> cp dprimer.ps ../junk                   Copy a file to parent directory.
olympus 64> cd ..                                   Move to the parent directory.
```

```
olympus 65> ls -l
total 120
-rw-r--r--   1 dclarke  staff      59573 Dec 29 13:54 junk
drwxr-xr-x   2 dclarke  staff        512 Dec 29 13:49 psfiles
olympus 66> chmod 000 junk                    Eliminate all permissions, even to me.
olympus 67> ls -l
total 120
----------   1 dclarke  staff      59573 Dec 29 13:54 junk
drwxr-xr-x   2 dclarke  staff        512 Dec 29 13:49 psfiles
olympus 68> rm junk  Without write permission, I get prompted if I wish to delete the file.
rm:  junk:  override protection 0 (yes/no)?  no                File is not deleted.
olympus 69> chmod 644 junk                                    Restore permissions.
olympus 70> rm junk    With write permission, I am not prompted when I delete the file.
olympus 71> touch newfile              The touch command will create an empty file.
olympus 72> ls -l                      The permissions of newfile are set by umask.
total 2
-rw-r--r--   1 dclarke  staff          0 Dec 29 13:59 newfile
drwxr-xr-x   2 dclarke  staff        512 Dec 29 13:49 psfiles
olympus 73> rm newfile                                        Remove newfile.
olympus 74> pwd
/disc6/home/dclarke/PHYS3437
olympus 75> cd                                                Move to the top directory.
/disc6/home/dclarke
olympus 76> setenv DISPLAY mm310.stmarys.ca:0.0
```

This command may be necessary if you are not logged into your "default device". The address `mm310.stmarys.ca` *is specific to the machine in room MM310; each machine will have its own unique address.*

```
olympus 77> te .cshrc &                The ampersand puts te into the "background".
```

`te` *will spawn a process, namely a textedit window (almost, but not quite the same as the* `textedit` *spawned from the toolbar), and until that process is finished (edits made, and window quit), your console will be frozen: i.e., no other Unix command can be input. By putting the process into the background, your console is released before* `te` *is finished, and you can have the textedit window active while at the same time issue new Unix commands in your console window. If you forget to add the ampersand, you can type* `<Control>Z`*, then* `bg<Return>`*, and this will put the process (your* `te` *window) into the background, and release your console. If for some reason you want to bring the process back into the foreground, type* `fg`*. Your console is then locked up until the process is killed, or put back into the background.*

```
[3] 1195                                  I changed umask in .cshrc from 022 to 077.
[3]    Done                  textedit .cshrc     Message generated when te is quit.

olympus 78> source .cshrc                 This makes the changes in .cshrc effective.
Executing system.cshrc on olympus ...
```

```
Thu Dec 29 14:04:22 AST 2005
/disc6/home/dclarke

olympus 79> cd PHYS3437
/disc6/home/dclarke/PHYS3437
olympus 80> touch newfile                          Now, create newfile again.
olympus 81> ls -l              Permissions on newfile now exclude group and world.
total 2
-rw-------   1 dclarke  staff          0 Dec 29 14:04 newfile
drwxr-xr-x   2 dclarke  staff        512 Dec 29 13:49 psfiles
olympus 82> rm newfile
olympus 83> cd
/disc6/home/dclarke
olympus 84> !te       The ! recalls the last command starting with the characters given.
te .cshrc &                                Change umask back to 022 in my .cshrc file.
[3] 1205
[3]    Done                  textedit .cshrc

olympus 85> source .cshrc
Executing system.cshrc on olympus ...
Thu Dec 29 14:06:41 AST 2005
/disc6/home/dclarke

olympus 86> cd PHYS3437
olympus 87> gnuplot                                An example of running gnuplot.

G N U P L O T
unix version 3.5 (pre 3.6)
patchlevel beta 293
last modified Thu Jun 27 23:02:31 BST 1996

Copyright(C) 1986 - 1995
Thomas Williams, Colin Kelley and many others

Send comments and requests for help to info-gnuplot@dartmouth.edu
Send bugs, suggestions and mods to bug-gnuplot@dartmouth.edu

Terminal type set to 'x11'              Default setting formats output for the screen.
gnuplot> set term post                    Instead, we want format to be postscript.
Terminal type set to 'postscript'
Options are 'landscape noenhanced monochrome dashed defaultplex "Helvetica" 14'
gnuplot> set output "file1.ps"                 Directs output to the named disc file.
gnuplot> plot [-pi:pi] [-1.1:1.1] sin(x)
```

Plots a sine wave with specified domain and range. Plot is dumped to file `file1.ps`

```
gnuplot> set term x11                                  Now format output for screen,
```

```
Terminal type set to 'x11'
Options are '0'
gnuplot> set output                                    direct output to screen,
gnuplot> plot [-pi:pi] [-1.1:1.1] sin(x)               and plot the same sine wave.
gnuplot> splot [-2:2] [-2:2] 3*x+2*y**2+4
```

Plot a 3-D plot with specified x- and y-domains.

```
gnuplot> quit
olympus 88> ls -l                    Note presence of file file1.ps, created by gnuplot.
total 28
-rw-r--r--   1 dclarke  staff      11279 Dec 29 14:07 file1.ps
drwxr-xr-x   2 dclarke  staff        512 Dec 29 13:49 psfiles
olympus 89> lp -d lj2 file1.ps                Print the file file1.ps to printer lj2.
```

5.4 Logging off, and a warning...

To log off, click the right mouse button on the "desktop" (background colour), and pull mouse down to "logout". Release the mouse key on "logout" and the system will prompt you to be sure you really want to log off; click yes if you are. If you are on a Sun workstation, you're done. If you are on a PC, you still have to log off the PC; all that logging out of the Unix session has done is to release Exceed™. Press `<Control><Alt><Del>` simultaneously, then click on "log off", to log yourself out of your PC account.

Under no circumstances should you leave your computer unattended without logging off first!

This includes stepping out of the lab for a few minutes just to go to the washroom. **Accounts found unattended will be closed unceremoniously.**