A *UNIX* Primer

David A. Clarke

Saint Mary's University, Halifax NS, Canada

dclarke@ap.smu.ca

September 2002; revised 12/05, 1/11

Contents

Preface

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

My challenge, then, is to get those in the second category up to speed fast enough to get them programming usefully without boring to death those in the first category, and to this end I have created this primer. For those with some computing experience, this document may serve as nothing more than a handy reminder for the one or two things that may have been forgotten. For the rest, this is what one needs to learn in order to function effectively in an undergraduate computational physics class. Believe it or not, this can be done in a matter of weeks, not months; one just needs to relax, be open-minded, and realise the world of computing is much more than just "point and click"!

I *may* know about 5% of what there is to know about *UNIX*, the subject of this primer, and these pages probably represent about 80% of that. As an *operating system*, *UNIX* is far more than what a programmer needs to know to function, and this primer is designed to filter out the essentials from the thousands of pages that comprise a typical exhaustive *UNIX* manual. And while these notes have been written with the needs of my own course in computational physics in mind, anyone else is free to use, distribute, and modify them as needed so long as they remain in the public domain and are passed on to others free of charge.

David Clarke
Saint Mary's University
January, 2011

Primers by David Clarke:

1. A *FORTRAN* Primer
2. A *UNIX* Primer
3. A DBX (debugger) Primer
4. A Primer on Tensor Calculus
5. A Primer on Magnetohydrodynamics
6. A Primer on *ZEUS-3D*

I also give a link to David R. Wilkins' excellent primer Getting Started with LaTeX, in which I have added a few sections on adding figures, colour, and HTML links.

A *UNIX* Primer

1 Overview

To a Computer Systems Administrator, an *operating system* (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, how information is sent to the screen, and so on. To the user, an OS consists of a set of allowed commands that may be typed in at the keyboard to accomplish rudimentary tasks. While this latter portion represents a very small part of the OS, it is probably 99% of what a typical user will ever see of an OS because it is this part one has to learn to use the computer; everything else is (thankfully) transparent. The purpose of this primer, then, is to introduce the small portion of the *UNIX* OS that allows a user to program a computer.

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, an e-mail, a computer program to solve a physics problem, *etc.*). 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" whatever applications (*e.g.*, your computer program, an internet browser, the window in which you are typing, *etc.*) you and other users may be using in and out of "memory" and on and off the processor in a way that is transparent to whomever may be logged on to the computer. When it was first introduced, *UNIX* was considered revolutionary and completely transformed how people managed computing resources.

UNIX was developed in the late 1960s 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 that ended up sticking. At the time, the prevailing OS at Bell Labs was called “`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 the 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*. Indeed, Mac OS X is based on Linux, a freely available version of *UNIX*, and all PCs these days are delivered with the ability to support Linux. At the time of this writing, our own computing systems in the Saint Mary’s Department of Astronomy and Physics supports a flavour of Linux called “`CentOS 5`$^{\text{TM}}$” (Community Enterprise Linux) which is publicly available free of charge and compatible with Red Hat Enterprise Linux.

UNIX is here to stay. Regardless of what flavour of *UNIX* a system may support, the beauty to a user is that each version looks and behaves essentially *identically* no matter the platform. What you learn about *UNIX* on your laptop is completely transferable to whatever supercomputer you may use down the road, and investing the time to learn *UNIX* now will give the student a skill they will use throughout their career as a professional scientist.

2 The Basics

This purpose of this document is to introduce the reader to the very basic fundamentals of *UNIX*. 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 listing the various *UNIX* websites here; they would be dated and possibly off-line as soon as this document is 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.

2.1 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 may be commands already known to the machine, or commands you create by generating "script files" (§2.2), 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, most of which are "hidden files" whose names are preceded with a period and which are not included in the file list when one submits a simple `ls` command (§2.4). **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 can remember for a *UNIX* command that may be too long to remember), and controls defaults for your mail, your prompt, where output goes, what permissions are given to new files, *etc.*

2.2 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, 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 §3.12), 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.

2.3 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 `/home/dclarke`, or equivalently, `~dclarke`. Your home directory will have an analogous name and is the area that you "own" where you can create and destroy files at will. As discussed in §3, 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 `PHYS2301`, `PHYS3210`, *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`; §3.12). A word of warning: the System Administrator can read all files—read 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!

2.4 Listing files (the `ls` command) and the *UNIX* prompt

The *UNIX* command to "go to" a desired directory is `cd` ("change directory"; §3.7) and to list the contents of a given directory is `ls` (list). All *UNIX* commands, including `cd` and `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 move to the directory

`~dclarke/stmarys/courses` on my laptop (which I've named `mytikas`), and then list the contents of that directory, this is what my terminal session looks like:

```
mytikas 1> cd ~dclarke/stmarys/courses
mytikas 2> ls
ASTR5510        PHYS1100        PHYS2301        PHYS3210        PHYS4790
ASTR5700        PHYS1101        PHYS3200        PHYS3500
ASTR5980-5981   PHYS2200        PHYS3201        PHYS4380
mytikas 3>
```

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
endif
```

at the very end of my `.cshrc` file, and the system substitutes the name of whatever machine I'm on (`mytikas` on my laptop, `mars.smu.ca` or `andromeda.smu.ca` on departmental machines) for `` `hostname` `` (note use of single "backquotes"). The cursor (a small triangle, a flashing underscore, or whatever the 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:

```
mytikas 3> ls -F
ASTR5510/       PHYS1100/       PHYS2301/       PHYS3210/       PHYS4790/
ASTR5700/       PHYS1101/       PHYS3200/       PHYS3500/
ASTR5980-5981/  PHYS2200/       PHYS3201/       PHYS4380/
mytikas 4>
```

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. Note also that any "hidden files" (those that start with a period; §2.1) are not included in the directory listing by typing `ls`. To see those as well, one must include the `-a` option; *e.g.* `ls -a`.

I can then direct the *UNIX* command to act only on one of the directories. For example, to list the contents of the directory `PHYS3210`:

```
mytikas 4> ls -F PHYS3210
assignments/    grades/         programs/       psfiles/
documents/      histograms/     projects/       syllabi/
mytikas 5>
```

These too are all directories since each has an appended slash. Note that 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.

Then, to see what is in the directory `syllabi`:

```
mytikas 5> ls -F PHYS3210/syllabi
syll02.latex    syll04.latex    syll06.ps       syll11.pdf
syll03.latex    syll06.latex    syll11.latex    syll11.ps
mytikas 6>
```

where all items are now files (no slash appended). Another useful option for `ls` is `-l`:

```
mytikas 6> ls -l PHYS3210/syllabi
total 360
-rw-r--r--  1 dclarke  staff   1689  7 Jan  2002 syll02.latex
-rw-r--r--  1 dclarke  staff   1665  2 Jan  2003 syll03.latex
-rw-r--r--  1 dclarke  staff   1791  5 Jan  2004 syll04.latex
-rw-r--r--  1 dclarke  staff   1999 31 Dec  2005 syll06.latex
-rw-r--r--  1 dclarke  staff  43854 31 Dec  2005 syll06.ps
-rw-r--r--  1 dclarke  staff   2842  6 Jan 11:27 syll11.latex
-rw-r--r--  1 dclarke  staff  27282  6 Jan 11:28 syll11.pdf
-rw-r--r--  1 dclarke  staff  88035  6 Jan 11:28 syll11.ps
mytikas 7>
```

With `-l`, more information is given for each file. The first columns (`-rw-r--r--`) indicate permission settings (who may read, write, and execute, and who may not; see §3.12). 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.

2.5 The "man" pages

For any command, you can find out much more than you may ever need or want 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.

3 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 §2.2, each *UNIX* command may be used as part of a script file which most new users start using early on.

3.1 `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.

3.2 `mkdir <directory name>`

This command creates a new directory with the specified name. 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 PHYS3210
```

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

3.3 `rmdir <directory name>`

This command removes an empty directory.

3.4 `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 /Volumes/disc7/dclarke scratch
```

will create a link called `scratch` that will appear as a subdirectory when all files are listed (`ls -F` will list it as `scratch@` with the `@` symbol identifying `scratch` as a link). Thus, typing `cd scratch` 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 quick access to other discs may be needed. For example, if you have an application that requires a great deal of disc storage, you can request access to "scratch disc space" which is not backed up, is purged once in a long while (with ample warning), but is semi-infinite in extent. 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.

3.5 `rm <file or link>`

This command 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 *` !

3.6 `pwd`

This command echoes the "present working directory", namely the directory in which you are currently working.

3.7 `cd <directory name>`

This command 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" and will search for that directory from the very top of the system's directory tree, and even above where your home directory is defined. Finally, the double dot (`..`) indicates "one directory up". Examples:

```
cd PHYS3210
```

changes directory to `PHYS3210`, and is successful so long as `PHYS3210` is a subdirectory of the present working directory.

```
cd /home/dclarke/stmarys/courses/PHYS3210
```

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 "`/home/`" bit. Thus,

```
cd ~dclarke/stmarys/courses/PHYS3210
```

is an equivalent command. Type:

```
cd ..
```

to moves up one directory, type:

```
cd ../..
```

to move up two directories, and type:

```
cd
```

to get to your top home directory. In my case, it is a short form for `cd /home/dclarke` or, equivalently, `cd ~dclarke`.

3.8 `mv <origin file> <target file>`

This command 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 PHYS3210/syllabi/syll11.latex ./syll11.latex
```

This command moves the file `syll11.latex` from directory `PHYS3210/syllabi` to the `pwd` directory (by virtue of the single dot in the place where the "target directory" goes) keeping the same name, and would be issued from the parent directory of `PHYS3210`. So long as the name of the file is to be left unchanged, the file name `syll11.latex` 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`.

Note that `mv` is what you use if all you want to do is change the name of a file. *e.g.*,

```
mv syll11.latex syllabus.latex
```

would change the name of the file containing the syllabus for the course from `syll11.latex` to `syllabus.latex`. This command as given would have to be issued from inside the subdirectory in which the file `syll11.latex` resides (*e.g.*, `~dclarke/stmarys/courses`). If I were "up one directory level" (and thus in `~dclarke/stmarys`), I'd type instead:

```
mv courses/syll11.latex courses/syllabus.latex
```

3.9 `cp <origin file> <target file>`

This command copies `origin file` to `target file`. Directory tree branches 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.

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

This command sends the postscript file `file name` to the specified printer. The Department does not maintain printers for undergraduate students. If you need to print off a file, you should go to a university site where printing services are offered. **NB: This command is for postscript files only. Sending non-postscript files to a printer using `lp` can cause a printer to stream out hundreds of pages of rubbish, so *be sure you know what you are sending off to the printer before printing!***

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

This command 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.

3.12 `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, use the `ls -l` command, as described in §2.4. For example, if I wished to examine the protections on this file (`uprimer.latex`), I would type: `ls -l uprimer.latex` to get 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 and on the departmental resources (`mars`, `andromeda`, *etc.*, but not my laptop `mytikas`), 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.

The `chmod` command allows you to change the permissions for owner, group, and world for any given file(s) or directories that you have created. It is this simplicity of design that also makes it relatively foolproof, and why *UNIX* systems are so blessedly difficult for hackers to hack into. They far prefer to try to break into the "wild west" that is MicroSoft's `WINDOWS`$^{\text{TM}}$ OS. Note that since you did not create your home directory (*e.g.*, `~dclarke`), you cannot change the permissions on it; only the System Administrator can do that. However, you can change the permissions for everything inside your home directory. Thus the command:

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

changes 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. If you do want to delete such a file at a future date, you must first change the permission to include write.

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, $\texttt{O} = 7 - \texttt{o}$, $\texttt{G} = 7 - \texttt{g}$, and $\texttt{W} = 7 - \texttt{w}$. 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).

3.13 Accelerants

`!` ("punch"):	typing `!cd`, for example, repeats the last command issued starting with `cd`.
`!!` ("double punch"):	repeats the last command issued.
`<esc>` ("escape key"):	completes the file or directory name in whatever command is being composed so long as enough characters have been typed to identify the file or directory uniquely.

3.14 `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.

Logging off by typing `lo` (a valid short form for `logout`) in a console window works so long as you logged on to the machine in that same window. If, as will likely be usual, you log on by creating a so-called “X-windows” link with your PC (§4.1), then the console window would be fired up *after* you have logged on to the machine, and thus typing `lo` in that window will not log you off. Instead, it will respond with the message:

`not a login shell`.

In that case, see §4.4 to log off the machine.

4 *UNIX* Practise Session

4.1 Accessing the A&P *UNIX* environment

In principle, one can access any machine connected to the internet from anywhere on the planet by using whatever logon protocol may be available. The possibilities are endless, and thus I shall restrict discussion here to the case where a student is using a departmentally controlled PC, such as those available in AT 325. Even from a public terminal room on Saint Mary's campus, accessing the Astronomy and Physics *UNIX* machines may be quite different from what is described here, and students wishing to access departmental systems from outside the department will have to consult the available technical support.

Thus, from an Astronomy and Physics PC:

1. `<Control><Alt><Del>` to log onto the PC. Enter your PC username and password in the appropriate fields and make sure "`DOMAIN`" is set to "`SMUAP`". This logs you into a `WINDOWS` environment, with various icons scattered around the screen.

2. To create a so-called "X-windows link" to one of the *UNIX* machines (by far the most convenient way to use these resources), left-click on the `START` icon at the lower left, and take the path: `START` → `All Programs` → `X-Win32 6.1` → `X-Win32`. This will clear most of the screen, leaving a box listing all the available *UNIX* resources.

3. Double left-click on `mars` or `andromeda`. It doesn't matter which; all your files will be equally visible from each.

4. A "`CentOS`" window is launched that pretty well fills the screen and prompts you first for your *UNIX* username, then password. Hit the `ENTER` or `RETURN` key after each. Note that your *UNIX* username and password may or may not be the same as your PC username and password, so be sure to type in your *UNIX* username and password here.

Welcome to *UNIX*!

4.2 The `CentOS` Window Environment

First, note that the `CentOS` window environment just launched is itself a regular PC window with the usual minimise/adjust/kill (_ □ ×) icons at the extreme top right. You can minimise this window down to the `WINDOWS` bottom toolbar at any time should you wish to access anything else on your PC.

In the `CentOS` window, there are three icons near the top left corner. The "computer" and "user's home" icons each give you browsing capability from different starting points. Double-left-click on any icon you wish to launch. The third icon is the usual trash bin.

While on *UNIX* you will likely spend the vast majority of your time using two applications: the terminal "console", and a "text editor". The former allows you to communicate with the computer using *UNIX* commands, while the latter allows you to create "ASCII files" such as computer programs, *UNIX* script files, LaTeX files, *etc.* ASCII is "ordinary text" (no highlighting or formatting characters common on Microsoft Word^{TM} 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.

To open a console window, right-click anywhere on the `CentOS` desktop and select `Open terminal`. This opens up a window within the `CentOS` window in which you can type *UNIX* commands at the *UNIX* "prompt", and is what you will need to perform the practise session in §4.3.

For a text editor, left-click on `Applications` at the left end of the `CentOS` toolbar (just under the MicroSoft window frame at the very top of your screen), and follow `Applications` → `Accessories` → `text editor`, the last being right at the bottom of the menu. This opens a `gedit` window (again, within the `CentOS` window) which is a completely intuitive text editor where you just need to point, click, and type. To reopen an existing file, just double-left click on the file icon, and a `gedit` window will open up with the file as you last left it. Alternately, you can just type `gedit <filename>` at the *UNIX* prompt of the console. With a bit of poking around, you can even discover how to render `gedit` as an icon that will live under the trashbin of your `CentOS` desktop, so you can click on this to launch a new `gedit` session, rather than having to browse for it under `Applications`.

4.3 Practise session on *UNIX*, including GNUPLOT

The following is a sample *UNIX* session that can be done within a *UNIX* terminal console (*e.g.*, §4.2), including an example of how to use `GNUPLOT`. `GNUPLOT` is a publicly available "cheap and cheerful" plotting package which you can download on your laptop. For further documentation, see http://www.gnuplot.info/faq/. Additionally, you can refer to the short primer by Henri Gavin.

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 are in *italics*. Not all of the computer-generated responses will be exactly as they appear on your screen (*e.g.*, the prompt, sizes of files, *etc.*), but most should be very close to what you see.

```
mars.smu.ca 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
mars.smu.ca 52> pwd
/home/dclarke
mars.smu.ca 53> ls -F
Desktop/         grad_poster/     mail/            spamass/
Mail/            grfx/            nmlst/           stmarys/
bin/             latex/           public_html/     theweb/
```

```
editor/         libs/           sci/            zeus/
mars.smu.ca 54> mkdir PHYS3210                             Making a new directory.
mars.smu.ca 55> dir                  I have alias dir "ls -F" in my .cshrc file.
Desktop/        editor/         libs/           sci/            zeus/
Mail/           grad_poster/    mail/           spamass/
PHYS3210/       grfx/           nmlst/          stmarys/
bin/            latex/          public_html/    theweb/
mars.smu.ca 56> cd PHYS3210
/home/dclarke/PHYS3210
mars.smu.ca 57> ls -F                          No files in the new directory yet.
mars.smu.ca 58> mkdir pdf_files   Create a subdirectory pdf_files within PHYS3210.
mars.smu.ca 59> cd pdf_files
mars.smu.ca 60> cp /home/dclarke/PHYS3210/pdf_files/* .
```

One can type ~ *instead of* /home/. *Thus,* ~dclarke/... *instead of* /home/dclarke/... *The asterisk means "everything", and the period means "to here" (i.e., the present working directory).*

```
mars.smu.ca 61> ls
dprimer.pdf    gnuplot.pdf    projects.pdf   uprimer.pdf
fprimer.pdf    lprimer.pdf    syll11.pdf
mars.smu.ca 62> ls -l
total 576
-rw-r--r--   1 dclarke  staff      71684 Jan  9 22:02 dprimer.pdf
-rw-r--r--   1 dclarke  staff     361771 Jan  9 22:02 fprimer.pdf
-rw-r--r--   1 dclarke  staff      32003 Jan  9 22:02 gnuplot.pdf
-rw-r--r--   1 dclarke  staff     347184 Jan  9 22:02 lprimer.pdf
-rw-r--r--   1 dclarke  staff      91711 Jan  9 22:04 projects.pdf
-rw-r--r--   1 dclarke  staff      27282 Jan  9 22:01 syll11.pdf
-rw-r--r--   1 dclarke  staff     178486 Jan  9 22:02 uprimer.pdf
mars.smu.ca 63> cp dprimer.pdf ../junk              Copy a file to parent directory.
mars.smu.ca 64> cd ..                                   Move to the parent directory.
mars.smu.ca 65> ls -l
total 120
-rw-r--r--   1 dclarke  staff      71684 Jan  9 22:02 junk
drwxr-xr-x   2 dclarke  staff        512 Jan  9 22:02 pdf_files
mars.smu.ca 66> chmod 000 junk                Eliminate all permissions, even to me.
mars.smu.ca 67> ls -l
total 120
----------   1 dclarke  staff      71684 Jan  9 22:02 junk
drwxr-xr-x   2 dclarke  staff        512 Jan  9 22:02 pdf_files
mars.smu.ca 68> rm junk
```

Without write permission, I get prompted if I wish to delete the file.

```
rm:  remove write-protected regular file 'junk'?  no             File is not deleted.
mars.smu.ca 69> chmod 644 junk                                    Restore permissions.
mars.smu.ca 70> rm junk
```

With write permission, I am not prompted when I delete the file.

```
mars.smu.ca 71> touch newfile         The touch command will create an empty file.
mars.smu.ca 72> ls -l                 The permissions of newfile are set by umask.
total 2
-rw-r--r--   1 dclarke  staff          0 Jan  9 22:03 newfile
drwxr-xr-x   2 dclarke  staff        512 Jan  9 22:02 pdf_files
mars.smu.ca 73> rm newfile                                         Remove newfile.
mars.smu.ca 74> cd                                       Move to the top directory.
/home/dclarke
mars.smu.ca 75> gedit .cshrc &
```

The ampersand (`&`*) puts* `gedit` *into the "background". Without it, your console is frozen and no other* `UNIX` *command can be input until* `gedit` *is quit. By putting the process in the background, your console is released before* `gedit` *is finished. The* `gedit` *window remains active while at the same time you can issue new* `UNIX` *commands in your console window. If you forget to add the* `&`*, you can type* `<Control>Z`*, then* `bg<Return>`*. This puts the process into the background and releases your console. If 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.*

```
[1] 1195                                I changed umask in .cshrc from 022 to 077.
[1]    Done            gedit .cshrc         Message generated when gedit is quit.
mars.smu.ca 76> source .cshrc            This makes the changes in .cshrc effective.
mars.smu.ca 77> cd PHYS3210
/home/dclarke/PHYS3210
mars.smu.ca 78> touch newfile                           Now, create newfile again.
mars.smu.ca 79> ls -l            Permissions on newfile now exclude group and world.
total 2
-rw-------   1 dclarke  staff          0 Jan  9 22:03 newfile
drwxr-xr-x   2 dclarke  staff        512 Jan  9 22:02 pdf_files
mars.smu.ca 80> rm newfile
mars.smu.ca 81> cd
/home/dclarke
mars.smu.ca 82> !ge         ! recalls the last command starting with the characters ge.
gedit .cshrc &                        I changed umask back to 022 in my .cshrc file.
[3] 1205
[3]    Done            gedit .cshrc
mars.smu.ca 83> source .cshrc
mars.smu.ca 84> cd PHYS3210
mars.smu.ca 85> gnuplot                             An example of running gnuplot.

        G N U P L O T
        unix version 4.0 patchlevel 0
        last modified Thu Apr 15 14:44:22 CEST 2004
        Copyright(C) 1986 - 1993, 1998, 2004
        Thomas Williams, Colin Kelley and many others

        This is gnuplot version 4.0.  Please refer to the documentation
        for command syntax changes.  The old syntax will be accepted
        throughout the 4.0 series, but all savefiles use the new syntax.

        Type 'help' to access the on-line reference manual.
        The gnuplot FAQ is available from http://www.gnuplot.info/faq/

        Send comments and requests for help to
                <gnuplot-info@lists.sourceforge.net>
        Send bugs, suggestions and modifications to
                <gnuplot-bugs@lists.sourceforge.net>

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 ... "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
mars.smu.ca 86> ls -l            Note presence of file file1.ps, created by gnuplot.
total 28
-rw-r--r--   1 dclarke  staff      33539 Jan  9 22:04 file1.ps
drwxr-xr-x   2 dclarke  staff        512 Jan  9 22:02 pdf_files
mars.smu.ca 87> gv file1.ps                     View the file file1.ps on the screen.
```

4.4 Logging off, and a warning. . .

To log off, left-click on `system` (on the `CentOS` menu bar near the top left) and follow the path `system` → `logout username`. The system will prompt you to be sure you really want to log off; click yes if you are. If the system doesn't log you of immediately, it may be because of a known bug in `CentOS` and its implementation here. If it hangs for more than a few seconds, click the red × at the extreme upper right corner of the `CentOS` window, and that will log you off.

Once logged off the *UNIX* machine, you will return to the PC environment where you will have to log off there separately; as far as your PC is concerned, all that logging out of the *UNIX* machine did was to quit `X-Win32`. 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.**