

Larry Herman
Jandelyn Plane
Gwen Kaye

September 1, 2009

Contents

1	Introduction	2
2	Getting started	3
2.1	Logging in	3
2.2	Logging out, and account security	3
2.3	Reauthenticating	4
3	Printing files	4
4	Xming	4
5	The Emacs text editor	5
5.1	Introduction	6
5.2	Cancel command	6
5.3	Creating a file and entering text	6
5.4	Editing a file	6
5.4.1	Basic cursor movement commands	7
5.4.2	Basic commands for deleting text	7
5.4.3	Deleting entire lines	7
5.4.4	Adding a line	7
5.5	Saving a file	8
5.6	Quitting Emacs	8
5.7	Editing an existing file	8
5.8	Syntax highlighting and automatic indentation	8
5.9	Word wrap in Emacs	9
5.10	Table of basic Emacs commands	9
6	Directories and basic file commands	10
6.1	Directories	10
6.2	Listing the files in the current directory	10
6.3	Constraints regarding filenames	11
6.4	Creating a new directory	11
6.5	Changing directories	12

6.6	Creating new files in the current directory	12
6.7	Displaying files' contents	12
6.8	Copying a file	12
6.9	Renaming (moving) a file	13
6.10	Deleting (removing) a file	13
6.11	Moving to a higher directory	13
6.12	Directory hierarchy	14
6.13	Removing (deleting) a subdirectory	14
6.14	Copying a file from another directory	14
6.15	Wildcard characters	15
6.16	Disk quota	16
6.17	Pipelines	16
7	Compiling and running C programs	16
7.1	Basic command-line compilation of a program	16
7.2	Output redirection	16
7.3	Supplying an alternative executable filename	17
7.4	Executing a program	17
7.5	Input redirection	17
7.6	Stopping a running program	18
8	Aliases	18
8.1	Adding aliases to the .aliases file	18
9	Using the online reference information	18
10	Advanced Emacs features	19
10.1	Line numbers and undo	19
10.2	Commands involved in moving text	19
10.3	Multiple windows	20
10.4	Searching for and replacing text	20
10.5	Getting help	21
10.6	Table of all Emacs commands	22
10.7	Conclusion	23

1 Introduction

This is a basic introduction to the UNIX operating system, particular to the OIT Grace UNIX Cluster and the UNIX facilities that you'll need to use in this course. This tutorial covers procedures for logging into the system, manipulating files and directories, compiling and running programs, and other information. If you know some of this information you don't need to read about it, but will need to read whatever information you don't know.

This tutorial is only an introduction, and hardly a comprehensive study of UNIX. UNIX is very powerful, and only the most important capabilities that you may need for doing your work in this course will be discussed. For those interested, further information on UNIX is available from OIT's website; click on the following link to visit the class webpage and see the relevant information there:

www.cs.umd.edu/class/fall2009/cm313/information.htm

Information in this tutorial that is to be typed exactly as it appears usually follows the UNIX prompt, which on the Grace cluster depends on which of the machines you're logged into, but will be of the format "baby:~:" (where "baby" represents and will be replaced by the name of the specific machine you're logged in to during any session). The prompt is printed by the system to indicate that it's waiting for you to type your next command in, but you don't actually type the prompt yourself. Explanations will appear in regular type. UNIX is case-sensitive, that is, it distinguishes between lowercase and uppercase letters, so you will need to type commands exactly as described, which is almost always in lowercase.

2 Getting started

2.1 Logging in

There are three steps involved in logging into the system: specifying the machine you want to use, giving your login id, and giving the password for that login id. Your login ID and password for your Glue account will be the same as your University directory ID and password.

Specifying the machine you will be using will vary from situation to situation. If you cannot immediately see how to connect to the machines using the directions your TA has given in discussion section, try coming to office hours. The Grace Cluster machines on which your Glue account works can be logged in to by using SSH (Secure Shell) to connect to "grace.umd.edu". There are two Solaris hosts in the Grace cluster and two Linux machines; SSH'ing to grace.umd.edu will just choose one of them at random. If you need to log in specifically to a Solaris machine you can SSH to solaris.grace.umd.edu, while if you need to log in specifically to a Linux machine you can SSH to linux.grace.umd.edu. For CMSC 313 you will need to use Linux, so you should **always** SSH to linux.grace.umd.edu. Note that code compiled on one type of Grace machine will not run on the other type.

Note that you have to use SSH to connect to any of the Grace machines, as telnet or other applications will not work. If you have a Windows computer and need an SSH client, a link where one may be downloaded from OIT is on the class webpage and can be reached via the following link:

www.cs.umd.edu/class/fall2009/cmsc313/information.htm

2.2 Logging out, and account security

After figuring out how to log in to the Grace systems you should next know, before anything else, how to log out. Don't ever leave yourself logged into your account from an unattended computer in a public computer lab where someone could get access to your account, even if you are just stepping out of the room for a minute. Lock the terminal if possible, or log out. It would be better to have to log back in when you return than to have someone steal or ruin your files.

Don't leave your session unattended anywhere else, even if someone you know is nearby to watch it for you. Supposed friends have victimized students in the past by stealing or ruining their data without their knowledge. You could be held liable by the University for damage caused by someone using your account, even without your knowledge, if they were able to access it through negligence such as described.

For an orderly exit, at the prompt type

```
baby:~: logout
```

(As above, "baby:~:" is the system prompt; you only type in the command "logout" after the prompt.) You can log into the system again if you just logged out.

2.3 Reauthenticating

The methods that the Glue and Grace systems use for authentication and security give users who are logged in something called Kerberos tickets, which expire after around 24 hours, meaning that if you have a login session that you keep active for longer than that, you won't be able to do many things in that session. If you try to access any files (yours or ours) you'll just get a message that permission is denied. You could log out and log back in, but typing the command "renew" will allow you to enter your password again and renew the authentication tickets for another 24 hours.

3 Printing files

The command to print one or more files, such as materials for this course, to the OIT printers, if the files are located on one of the OIT systems is:

```
baby:~: qpr -x duplex -q avw names-of-files-to-print
```

You can transfer files to the Grace systems from elsewhere (such as your own computer) using the same SSH client you use to log in (use the file transfer window). Note that you have to have an OIT print account in order to use the OIT printers, and pay for printouts by the page. Information on getting a print account and using pay-for-print can be found on OIT's webpage, linked to from the class webpage, which may be accessed at the following link:

www.cs.umd.edu/class/fall2009/cmsc313/information.htm

If you want to print on only one side of the page, omit the "-x duplex" option from the command above:

```
baby:~: qpr -q avw names-of-files-to-print
```

If you really want to save paper, you can use the mpage utility to print your files reduced so that two pages fit on each side of each piece of paper. The command to do so is:

```
baby:~: mpage -2ft -Pavw names-of-files-to-print
```

Note there is no space between "-P" and "avw". Since this command will print two pages per each side of a piece of paper, and print on both sides of the page, therefore four pages of each file will be printed on each piece of paper.

To use mpage to print two pages per each side of paper, but only single-sided, omit the 't' in the options above:

```
baby:~: mpage -2f -Pavw names-of-files-to-print
```

Printouts sent using commands as given above will be sent to the OIT pay-for-print printer in the AVW Glue lab, which is AVW 1120. Several other Glue labs on campus have printers; their locations may be found at:

www.helpdesk.umd.edu/systems/lpccr/connectivity/labs/180

In the commands above, replace "avw" with the queue name of the desired printer from this webpage to use one of the printers in the other labs.

4 Xming

Unlike Microsoft Windows, the window system in UNIX isn't an integral part of the operating system; it's a separate program, which means that different window managers are available. The window system that is prevalent in UNIX is X Windows. If you're logged into a UNIX host using a Mac or UNIX/Linux computer, your machine is already running an X server, meaning that applications that

want to run in separate windows can do so. If you connect to a UNIX host using a computer that isn't running an X server (such as Windows), then many applications that want to run in separate windows just won't work, or may work in a limited fashion. However, implementations of X Windows are available for Windows, one of which is Xming. As long as you're using a high-speed connection, installing Xming will allow applications to run in their own windows, for example the Emacs text editor (explained in the next section) will open up in its own window, allowing you to continue using the terminal window for typing commands while you're editing files. If you're using a Windows computer to connect to the Linux Grace hosts and have high-speed internet access, using Xming is highly recommended.

Xming for Windows can be downloaded using the link below. You don't have to install it now, but if you do you'll find Emacs (explained below) will be much more convenient to use.

http://sourceforge.net/project/showfiles.php?group_id=156984

You only have to download Xming itself and run the installer. Once it's installed, to run it:

- First enable X11 tunneling— start your Windows SSH program and use it to connect to linux.grace.umd.edu.

Go into Edit, Settings, Tunneling (under Connections), and click “Tunnel X11 connections”. Click OK, and then File, Save Settings, so this setting is saved. You only have to do this step once.

- Then quit the SSH program, since Xming has to be running first for things to work.
- Then run Xming by double-clicking the desktop icon.
- Then (re)run the Windows SSH program.
- Now when you connect to the Grace hosts and run any application that would cause a window to open, it'll pop up as a separate Windows window.

Just run Xming by double-clicking on its icon every time you start Windows. If you ever use the SSH program to log into the Grace systems without first running Xming, you'll have to first disable X11 tunneling, otherwise you'll get an error message.

5 The Emacs text editor

If you've used an integrated development environment such as Eclipse to write programs, you will find the typical development process in UNIX is a bit different. Although a number of more sophisticated program development tools are available in UNIX these are beyond the scope of this course, so just the basic program development utilities are described here.

To write and run a C program in UNIX you would first use a text editor to type in the program code and save it to a file. A text editor is similar to a word processor, and is used to create and save text files, such as C programs. You would then run the C compiler, which is a separate program. The compiler produces an executable program as a file in your directory, which you then run or execute. This is called the edit-compile-execute cycle, and this tutorial addresses in detail what has to be done in all three steps. If errors occur at either of the last two steps (the program fails to compile, or it compiles and runs but doesn't work correctly) you would have to figure out what's wrong and use the text editor again to make changes to it, and save it again so you can compile again.

Various text editors are available for use with UNIX; this tutorial explains basic use of Emacs. The best way to learn these commands is to experiment and practice with them. If you happen to already know another full-featured UNIX text editor, such as vi, feel free to use it instead for entering

your projects. (You are strongly discouraged from using a simple text editor such as Pico for editing program text.)

If you're running Emacs in such a way that it opens up in its own window (for example, if you are running Emacs while logged in from a UNIX workstation, or from any machine that is running an X server such as Xming) then you can use commands in its menu system by clicking on them with the mouse. This should be sufficiently self-explanatory that it won't be discussed here further. However, at times it may be necessary to use Emacs in a situation where the menu commands can't be accessed using the mouse, so below we'll explain how to use the most common Emacs commands just using keystrokes.

5.1 Introduction

The Emacs editor uses modeless editing, meaning that commands **always** use a special key followed by a letter or other symbol. For example, CONTROL-f, that is, holding down the CONTROL key and pressing 'f' at the same time, will move the cursor forward one character. Unless preceded by a special key, ordinary characters typed are simply inserted in the document being edited. In the following discussion C indicates the CONTROL key while M indicates the ESCAPE key (also known in Emacs as the META key). A command like C-f indicates that while holding CONTROL you should press the 'f' key. On the other hand, M-v means you should press ESCAPE and release it, and then press 'v'.

5.2 Cancel command

Before you start Emacs you should know the cancel command, which is C-g. This should cancel any pending Emacs command. If you type the wrong keys and aren't sure what Emacs is doing, or perhaps what it's waiting for you to type, C-g should cancel it, and leave Emacs back in its normal mode.

5.3 Creating a file and entering text

You're encouraged to start Emacs now and try the commands as they're described. For example, start Emacs with the name of a file to be created (suppose you want to use the name "tempfile"):

```
baby:~: emacs tempfile
```

If Emacs doesn't start (especially you are logging in from a PC running Windows and aren't using Xming), and an error message about a DISPLAY environment variable appears instead, run Emacs with the "-nw" option (e.g., "emacs -nw tempfile").

When Emacs begins it will start with an essentially blank screen. The second-to-last line is the status line, which gives the name of the file you are editing. The very bottom line is where you will have to enter any necessary information that Emacs may ask you. If you start Emacs and begin to type a few characters, as you enter the first character "**" will appear in the left of the status line near the bottom of the screen. This indicates that you have made changes in the file since it was last saved (or, if it's a new file, it has never been saved). Now type a few characters and try saving your work and exiting Emacs, by typing C-x C-s to save (hold down the CONTROL key and press x, then hold down the CONTROL key and press s). To exit Emacs then type C-x C-c, and you should find yourself back at the UNIX prompt.

5.4 Editing a file

The information below presents the basic commands for modifying or editing files. Besides the arrow keys and page-up/page-down keys, Emacs has other key combinations you can use for basic cursor

movement if desired.

5.4.1 Basic cursor movement commands

forward one character	the right-arrow key or C-f
backward one character	the left-arrow key or C-b
previous (up) one line	the up-arrow key or C-p
next (down) one line	the down-arrow key or C-n
one whole page up	the Page Down key or M-v
one whole page down	the Page Up key or C-v
beginning of current line	the Home key or C-a
end of current line	the End key or C-e
beginning of file	M-<
end of file	M->

Experiment with these commands so that you will become familiar with them; for instance, type M-> and see that the cursor moves to the bottom of the file you're editing.

5.4.2 Basic commands for deleting text

Deleting a character forward means the character under the cursor is deleted, and all the characters on the rest of the line (from the next character until the end) are moved one position to the left. If the cursor is located after the last character on the line, delete character forward will cause the next line to move up and be joined to the end of the current line, right after the cursor position.

Deleting a character backwards means the character to the left of the cursor position is deleted, and all the rest of the line is shifted to the left (including the character under the cursor and the cursor itself). If the cursor is located at the beginning of a line, delete character backward will cause the current line to move up and be joined to the end of the previous line.

delete character forward	C-d
delete character backward	the backspace key on the keyboard, or C-h, which also does this
delete characters to end of line	C-k

5.4.3 Deleting entire lines

C-k will remove or delete all of the current line after the current cursor position, so if you move to the beginning of a line and press C-k it will delete the entire line, but a blank line remains. Type C-d, or C-k again, to remove the end-of-line marker and remove this blank line. If you want to keep part of the beginning of a line but delete the last part you can use it to do so; just move the cursor to the end of the part you want to keep and press C-k.

5.4.4 Adding a line

To add a line below an existing line, move the cursor to the end of the line before where you want to add the new one and hit RETURN to open up a new line. Alternatively, you can move the cursor to the beginning of the line where you want to add a new one and type C-o to open up a new line after the current cursor position.

5.5 Saving a file

So far any work you have done hasn't been written to a file. It is all stored in Emacs' memory. To write the current contents of the text stored in Emacs to the file whose name you typed when starting Emacs, that is, "tempfile" above, type `C-x C-s` again, as you did earlier.

Very important: be sure to save your work **often** while you are working. Every programmer has at least one horror story of typing a program for hours and just as it's finished the power goes out or the computer locks up for some reason, and all their work is lost. A good habit to get into is to type `C-x C-s` every several minutes while you are working. Also, in order to avoid inadvertently deleting or incorrectly changing the only copy of a file it's essential to keep backup copies of important files. A good practice is to save copies of coursework you're working on in a different directory or with a different filename, using the UNIX commands explained below.

5.6 Quitting Emacs

If you type `C-x C-c` to quit Emacs and you have made any changes to your document, Emacs asks you (in the very bottom line) whether you want to save your file. Answer by pressing 'y' or 'n'. If you are editing more than one file (in different windows, as described below) and several of them have been changed, Emacs will ask you about saving each one. Emacs will then ask

```
Modified buffers exist; exit anyway (yes or no)?
```

If you're sure you want to quit, type "yes" and press return.

5.7 Editing an existing file

If you have exited editing a file like "tempfile", to return to it for modification, give the command:

```
baby:~: emacs tempfile
```

When you edit an existing file and update it, Emacs will save the original version as a backup copy by renaming it. The backup will have the same name as the original file but with a ~ character at the end. For example, if you make changes to a file named `proj1.c`, Emacs will save the original version of the file as `proj1.c~`. If you edit the file again and make more changes Emacs will again save the current version as `proj1.c~`. If you ever find that you've incorrectly changed a file and want to revert to its previous state you can use the backup file, after renaming it to prevent Emacs from overwriting it (when you next edit the original file).

5.8 Syntax highlighting and automatic indentation

For some programming languages Emacs has syntax highlighting built in, in which different program elements can be displayed in different colors. For other languages you can enable this with just a little bit of customization if desired. To turn on syntax highlighting while editing a C program file you can press `ESCAPE` and then 'x', causing the cursor to move to the status line at the bottom, of the screen, waiting for you to enter an Emacs command, and type `font-lock-mode`. This will turn on syntax highlighting for that editing session for that file only. If you want to always enable syntax highlighting while editing any C files add the following line to the end of the file `.emacs` in your home directory:

```
(add-hook 'c-mode-hook 'turn-on-font-lock)
```

Include the line exactly as shown. The `.emacs` file is read by Emacs every time you run it, so once you learn more about Emacs you can put various customizations that you like there.

By default, when editing plain text, Emacs does not have the word wrap feature that you may be accustomed to from using word processors enabled, so you must press the return or enter key at the

end of each line. However, if you're editing C program text Emacs can indent it for you automatically. If you press `C-j` at the end of a line **instead** of pressing return, the cursor will advance to the next line and properly indent it, based on the structure of the program as entered so far. Alternatively, you can just press return at the end of a line of program code, but press the tab key before starting to type the next line, and Emacs will indent correctly in this case as well.

5.9 Word wrap in Emacs

Emacs customizes its behavior depending upon what type of file you're editing. When you start typing C programs you'll find that, while editing programs, Emacs does not have the word wrap feature that word processors typically have, since you know that programming languages often require that some statements or constructs can't be broken at arbitrary places. Therefore if you need information to be on the next line, you must explicitly press the enter or return key. Since there is no word wrap, just adding spaces until the cursor appears to be on the next line won't work, because what you'll actually have is a very long line with a lot of spaces in the middle.

5.10 Table of basic Emacs commands

If you're a new Emacs user you can print this table and keep it handy while you're first using Emacs.

`C-` means hold down the CONTROL key while pressing the next key mentioned.

`M-` means you should press the ESCAPE key, release it, and then press the next key mentioned.

Cancel command:

`C-g`

Basic cursor movement commands:

forward one character	the right-arrow key or <code>C-f</code>
backward one character	the left-arrow key or <code>C-b</code>
previous (up) one line	the up-arrow key or <code>C-p</code>
next (down) one line	the down-arrow key or <code>C-n</code>
one whole page up	the Page Down key or <code>M-v</code>
one whole page down	the Page Up key or <code>C-v</code>
beginning of current line	the Home key or <code>C-a</code>
end of current line	the End key or <code>C-e</code>
beginning of file	<code>M-<</code>
end of file	<code>M-></code>

Basic commands for deleting text:

delete character forward	<code>C-d</code>
delete character backward	the backspace key on the keyboard, or <code>C-h</code> , which also does this
delete characters to end of line	<code>C-k</code>

Saving and quitting:

save file	<code>C-x C-s</code>
quit Emacs	<code>C-x C-c</code>

Several additional powerful and convenient features of Emacs are described in Section 10 at the end of this tutorial, which can make editing files much more convenient, but since the commands described in this section are sufficient to get started with we'll defer discussion of those till later, after basic UNIX commands have been explained.

6 Directories and basic file commands

6.1 Directories

Directories are where files are stored (you can think of them as analogous to folders in Windows or Mac terminology), and UNIX's directory organization is hierarchical. At the top of this hierarchy is the root directory, designated by a slash (/). Under the root are other directories that contain information such as often-used utilities, and user and system accounts. At any time whichever directory you are currently examining (or located in) is called the current directory. You can move from one directory to another in the UNIX filesystem analogous to how you can move from room to room within a building. When you first log in your current directory is your own account's home directory. You can only change or move to directories for which you have been given permission.

Subdirectories allow you to organize files by grouping related ones. For instance, you can store all the files for your first project in a subdirectory named, for example, proj1, all the files for your second project in a subdirectory named proj2, etc. Even if some of the files you use for Projects #1 and #2 have the same names there won't be any conflict and you can have complete copies of both projects in your account at the same time, because files in different directories can have the same names.

In UNIX a single period (.) refers to the current directory, and two adjacent periods (..) refers to the directory one level above (closer to the root) the current directory. The tilde symbol (~) refers to your home directory, and the ~ followed by some other account ID refers to that account's home directory. Your home directory corresponds to your Glue login id and will not change; you will be in your home directory as you log in each time. The current directory corresponds to your home directory as you first log in, but the current directory will change each time you move to another directory; things in the current directory will be assumed as the default when you give a command.

A pathname is a description of where a file is located, including the directory names that form a path to the directory where the file is located. The slash (/) designates the root directory, and is also used to separate directory names in a pathname. For instance, the pathname

```
/afs/glue/class/fall2009/cmsc/313/0101/public/testfile
```

refers to a hypothetical file named "testfile" located in the "public" subdirectory of the "0101" subdirectory of the "313" directory, etc. Files that are not in the current directory are sometimes specified using a pathname like this showing their location in relation to the root directory. They are also often specified relative to the current directory or relative to a home directory (yours or another user's), using ~.

6.2 Listing the files in the current directory

List just the names of files and subdirectories in the current directory by typing

```
baby:~: ls
```

Any file whose name begins with the period character is a "hidden" file, and not ordinarily listed by ls. Your account contains several of these files, including some that are used when you first log in. To list all your files listed by ls plus any hidden files you have to supply an option to the ls command;

options modify the default behavior of commands. Options in UNIX are preceded by a minus sign. To list all your files type

```
baby:~: ls -a
```

To get more information about the files and directories, type:

```
baby:~: ls -l
```

For each file or directory in the current directory, there will be a line that looks like this.

```
-rw-r--r--    1 gwen    osl          2565  Nov 16 2007 office
drwxr-xr-x    2 gwen    osl           512   Jan  5 2007 prereg
```

The following columns are of interest:

1. The first character 'd' indicates this is a directory, '-' indicates it's a file.
2. The next nine characters specify the access permissions for this file or directory, using three sequences of three characters each to indicate permissions. The first three of these characters indicates the type of access that the person who created the file has for it: read (r), write(w), and execute(x). A dash in place of any one of these letters signifies that the user does not have that type of permission for the file or directory. A user needs execute permission to change or move into a directory. The next six characters refer to permissions other people have to access files, that isn't relevant for this course.
3. The third column specifies the name of the owner of the file, which will be your login ID for the files you create.
4. The fifth column has the size of the file, in bytes.
5. The next three columns have the date of creation or of last modification of the file or directory.
6. The last column gives the name of the file or directory.

6.3 Constraints regarding filenames

When creating files (or directories) of your own, any character can be part of a filename (or a directory name) subject to the following constraints. If a period (.) is used as the first character of a filename or directory name, the file or directory will be hidden. If / appears in the middle of a filename what comes before it will be interpreted as a subdirectory name, which probably isn't what was desired. Avoid file or directory names with - or # as the first character, or ~ as the last character. Since - is used to identify options for commands, a filename with - will often be interpreted by commands as an option instead of a filename. File or directory names that begin with # are considered temporary files and may periodically be deleted by the system. Filenames ending in ~ are treated as backup files by Emacs, and may be therefore be overwritten.

6.4 Creating a new directory

To create a new directory called, say, "examples" within the current directory, type

```
baby:~: mkdir examples
```

You can list your current directory, using `ls`, to see that this new directory has been created.

6.5 Changing directories

Changing to a directory means making that named directory the default for commands given. This named directory then is called the current working directory.

Change to this new directory:

```
baby:~: cd examples
```

You will see that there are no files in this directory at this point.

```
baby:~: ls
```

While you are located in the examples subdirectory, the files in your home directory can be listed by typing any one of the following:

```
baby:~: ls ..
baby:~: ls
baby:~: ls ~loginID
```

(where as above *loginID* represents your own Glue user ID or login ID).

6.6 Creating new files in the current directory

Using a text editor (such as Emacs), create two small files in this directory. The first file named “file1” should contain the following text:

```
This is file1.
```

The second file named “file2” should contain:

```
This is file2.
```

List the files in this directory in the way you did above. You should see file1 and file2 there now.

6.7 Displaying files’ contents

The cat command displays the entire contents of a file, so you can see file2’s contents by typing:

```
baby:~: cat file2
```

If the file being displayed contains more than a screenful of text, the more command is preferable, since it displays the file’s contents one page at a time, pausing after each page.

```
baby:~: more file2
```

After each page is displayed pressing the space bar (when the “--More--” prompt is seen) causes the next page to be displayed, while pressing return or enter causes only the next one line to be shown. Pressing the ‘q’ key quits displaying the file’s contents.

The less command does the same thing as the more command, but has additional capabilities (less is more in this case); you’re encouraged to just use less in place of more in commands like that above.

6.8 Copying a file

Now copy the contents of file1 to a new file that will be called file3.

```
baby:~: cp file1 file3
```

List the files in the directory. file3 should have been added to file1 and file2 in the listing.

Look at file3 to see what it has in it, using the cat command:

```
baby:~: cat file3
```

Since you're still located in the subdirectory you created of your home directory, to make a copy of file1 in your home directory with the same name you can use any one of the following:

```
baby:~: cp file1 ..
baby:~: cp file1 ~
baby:~: cp file1 ~loginID
```

If the second word after the cp command is a directory name (not a filename) then the file will be copied to that directory, using the same name as it presently has.

To make a copy of file1 in your home directory with a different name, such as "testfile", you can use any of the following:

```
baby:~: cp file1 ../testfile
baby:~: cp file1 ~/testfile
baby:~: cp file1 ~loginID/testfile
```

If the second word after the cp command is a filename with another directory's name in front of it, then the file will be copied to that directory, with that name. The first of these commands shows where you want testfile to be located relative to the current directory (i.e., one directory above the current one). The second and third show where you want testfile to be located relative to your home directory. If your login ID (home directory) was named student123 and you knew that it was located, for example, in the directory /homes, you could also make a copy of file1 from the current directory to your home directory with the name testfile using the command "cp file1 /homes/student123/testfile", but since the full path name is extra work to type there isn't much advantage to doing so.

6.9 Renaming (moving) a file

"Move" file1 by giving it the new name file4:

```
baby:~: mv file1 file4
```

Look at file4 to see what it has in it:

```
baby:~: cat file4
```

List using ls. Now the directory should show file2, file3, and file4. Obviously, the difference between a copy and a move is that copy makes an additional copy of the file while move "moves" the file's contents to another file. It should be noted that there is not actually a move, but a renaming of the file. Since file1 was moved to file4, file1 is no longer in the directory.

6.10 Deleting (removing) a file

```
baby:~: rm file3
```

Now list. The directory should show just file2 and file4.

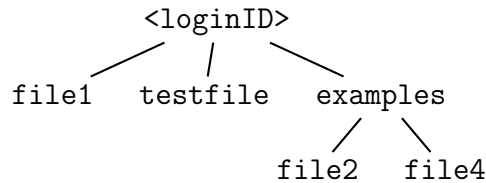
6.11 Moving to a higher directory

```
baby:~: cd ..
```

Now list. You are now back in the directory that is one level higher in the directory tree, in this case the original directory.

6.12 Directory hierarchy

A diagram of the hierarchical structure of your account now looks like this.



But actually your “account” is just a subdirectory in the directory of accounts on the system. To see the path, that is, the chain of directories leading from the root to your account, type the command to print the working directory:

```
baby:~: pwd
```

The first slash in this listing of directories represents the root. The names that follow represent directories leading directly from the root to your account directory. The slashes other than the first are delimiters. `cd ..` will change to the next higher directory, and

```
baby:~: cd ~loginID
```

will return to your own directory (or as a shortcut you could simply type `cd`, which returns you to your home directory.)

Notice that there are four ways to specify the location of a file or directory, depending upon the beginning characters of the path used to refer to it:

/	specifies the path to a file or directory beginning at the root or top of the filesystem or directory structure
..	specifies the path to a file or directory beginning at the parent directory of the current directory
~ or ~loginID	specifies the path to a file or directory beginning at your own home directory (the first form), or at the home directory of the user <i>loginID</i> (the second form)
anything else	specifies the path to a file or directory beginning at the current directory

6.13 Removing (deleting) a subdirectory

The `rmdir` command deletes a subdirectory. A subdirectory that contains any files can't be deleted. All files contained must be deleted before a subdirectory can be removed.

6.14 Copying a file from another directory

You may, if the permissions are set correctly, copy files from another account or directory. Your instructor may have given you the name of a public directory for your class where you can access certain materials that may be placed there. Try listing the files in that directory, for example:

```
baby:~: ls -l /afs/glue/class/fall2009/cmssc/313/0101/public
```

Say there is a file named “tempfile” in this directory that you want to copy to your home directory. You can either do this if you are located in your home directory, or if you are located in the directory `/afs/glue/class/fall2009/cmssc/313/0101/public`.

The first way, assuming you are already located in your home directory:

```
baby:~: cp /afs/glue/class/fall2009/cmssc/313/0101/public/temp .
```

or

```
baby:~: cp /afs/glue/class/fall2009/cmsc/313/0101/public/temp temp2
```

The second form above copies the file temp to your home directory and names it temp2, while the first form copies it to your home directory keeping its original name temp.

The second way to copy the file would be to change your location to the directory where the file is (/afs/glue/class/fall2009/cmsc/313/0101/public), and copy the file to your home directory from there:

```
baby:~: cd /afs/glue/class/fall2009/cmsc/313/0101/public
baby:~: cp temp ~
```

or

```
baby:~: cd /afs/glue/class/fall2009/cmsc/313/0101/public
baby:~: cp temp ~/temp2
```

Again, the first form retains the name temp for the copied file, while the second form makes a copy of the file with the new name temp2.

Experiment by copying files using both of these methods, using different filenames. Once you have a few files in your directory or even in some subdirectories, you can also experiment with the other UNIX commands listed above, such as those to rename files, view files, etc.

Using the -r option with cp copies a subdirectory along with all the files it contains, from one location to another.

6.15 Wildcard characters

Wildcard characters can be used in file names to stand for other characters. Using them can often make commands easier to type, or can allow several separate commands to be replaced by one combined command. UNIX has a powerful set of wildcard characters and methods for specifying file names; we will present only the two most useful here, which are the characters ? and *.

A ? in a file name appearing in a command stands for any other character, while an * in a file stands for any number of zero or more characters. To illustrate their use, suppose your directory contains the following files:

answer2.txt	answer5.txt	hw1.ans	hw3	program2	proj3
answer3.txt	answer6.txt	hw2	hw3.ans	proj1	proj4
answer4.txt	hw1	hw2.ans	program1	proj2	txt

The command “ls hw?” lists all files whose names are “hw” followed by any other character, in other words, in this case the files hw1, hw2, and hw3.

The command “ls hw*” lists all files whose names begin with “hw” and are followed by any other characters, in other words, the files hw1, hw1.ans, hw2, hw2.ans, hw3, and hw3.ans.

The command “ls *txt” lists all files whose names end in “txt”, in other words, the files answer2.txt, answer3.txt, answer4.txt, answer5.txt, answer6.txt, and just txt.

The command “ls proj?” lists the files proj1, proj2, proj3, and proj4.

The command “ls pro*” lists program1, program2, proj1, proj2, proj3, and proj4.

The command “ls *2*” lists all files whose names contain a 2, which is answer2.txt, hw2, hw2.ans, program2, and proj2.

Other commands besides ls can use wildcard characters, as in the following:

baby:~: cp ~fred/*.c .	copy all files whose names end in “.c” from the user fred’s directory to your own
baby:~: rm hw?	delete hw1, hw2, and hw3

Be very careful when using wildcard characters with the `mv` and `rm` commands, which can delete or overwrite a file. If you're not sure whether a command will work correctly, either don't try it, or make backup copies of your files under different names, or in a subdirectory, ahead of time. It is a very good idea to always keep backup copies of any important files, in case you accidentally delete or change them.

6.16 Disk quota

Every account has a limit to the amount of information that can be stored. To see what your limit is and how much space you have available, type the command `quota`. This lists your current disk usage as well as your quota. If you see, during the semester, that you are getting close to your quota you will need to delete any unnecessary files.

6.17 Pipelines

UNIX pipelines are used to run two commands, with the output of one command being used as the input for another command. The simplest form of a piped command is:

```
baby:~: command1 | command2
```

This executes *command1* and instead of sending its output to the screen it becomes the input for the command *command2*. For example:

```
baby:~: ls | more
```

Here the list of files in the current directory is displayed one screen at a time.

7 Compiling and running C programs

7.1 Basic command-line compilation of a program

To compile a program written in C, first note that it must be in a file whose name has the extension `.c`. We will discuss in lecture and discussion what a C program looks like and what information it must contain, as well as how to use a text editor to type a C program into a file in your account, if you don't know already. The name of the C compiler we will be using on the Grace Cluster is `gcc`, and if you had a C program in a file named `test.c` it could be compiled like so:

```
baby:~: gcc test.c
```

7.2 Output redirection

If your program had any mistakes you would see error messages when you tried to compile it. Using Emacs (or another text editor), you would have to find the causes of these error messages, correct them, and recompile. Sometimes you will have more than a single screenful of error messages. As it's difficult to read the error messages in this case, it will be helpful to send a list of these messages to a file called, for example, `myerrors` by expanding on the command to compile:

```
baby:~: gcc test.c >& myerrors
```

The `>&` tells the operating system that the results of the command to the left (that compiles the program) are not to be printed to the screen, but are instead to be written to the file name that appears to the right. You can then look in `myerrors` to get a list of your errors. If there are no compilation errors the file will be empty.

If you wanted to see the errors listed on the screen one page at a time you could alternatively type

```
baby:~: gcc test.c |& more
```

This runs the command to the left of the `|&` symbol, sending all its output to the command “`more`” instead of sending it to the screen or to a file. The `|&` form sends not only the regular output of the compilation command on the left of the pipe symbol as the input of the `more` command on the right, but includes error messages as well. The “`more`” command, discussed in Section 6.7 above, displays information one screenful at a time; in this case the information is the warnings and errors produced by the compiler, rather than a file’s contents (the compiler doesn’t usually print any output except error messages).

Once you correct all of your errors and you have a clean compile, the executable file created from your program will be placed in the current directory and named “`a.out`”. To see this file in your directory, type:

```
baby:~: ls -l
```

You will notice that `a.out` has executable permission since it is an executable file.

7.3 Supplying an alternative executable filename

If you use the `-o` option you can tell the compiler to put the executable program in a file with a name other than `a.out`. For instance,

```
baby:~: gcc test.c -o test
```

puts the executable program in a file called “`test`”. There is a big advantage to using an executable file name other than `a.out`. Only one file in a directory at a time can have the name `a.out`, so if you want to run several different programs you would have to compile each one first, but if you compile them and give them different executable names then they can all be present at the same time and you can run them afterward without having to compile again.

You execute a program in UNIX just by typing its name, such as

```
baby:~: test
```

7.4 Executing a program

You execute a program just by typing its name, such as

```
baby:~: test
```

7.5 Input redirection

If a program reads a lot of information and needs to be executed many times it would be too time-consuming to type the information each time the program was run. You can instead type the input data into a file using a text editor and run the program so that every time it reads information it doesn’t look to the keyboard for data to be typed in, but instead reads the next value from this data file. To run the above program so it reads all of its input from the file “`my-data`”, you would type:

```
baby:~: test < my-data
```

When a program is run with input redirected it reads its input without it being displayed upon the screen; however, all output will appear. Both input and output redirection can be used with the same command, as in “`test < infile > outfile`”.

7.6 Stopping a running program

If a program of yours doesn't seem to be working right you can always stop it by typing control-c (hold down the control key and press c). Most UNIX commands can be stopped as well by typing control-c. (Do not use control-z, which may appear to work but has an entirely different effect, and keeps a program running and using system resources.)

8 Aliases

UNIX allows you to create a shorter alias to facilitate typing long or complicated commands. A command like:

```
alias word "replacement"
```

will cause *word* when typed in to be replaced by *replacement* (that can consist of several words if desired).

8.1 Adding aliases to the .aliases file

One special hidden file, named “.cshrc”, contains commands that are automatically executed every time you log in. Any commands placed there will therefore be executed and set up every time you log in. On Glue and Grace systems, the .cshrc file will automatically read any aliases in a file in your home directory named “.aliases”, if it exists.

Now that you know what aliases are, if you want to create any of your own as shortcuts for typing long commands you can edit your .aliases file using a text editor, and add the desired aliases at the bottom. For instance, many users like to alias `ls` to “`ls -F`”, which makes it easy to identify the types of files when they are listed, because different types of files are marked with a special character after their name (executable files' names end in `'*`', directory names end in `'/'`, etc.). Try this option and if you like it you can add it to your .aliases file. You can add any other aliases you find useful.

A sample .aliases file with two added aliases might be:

```
# This file is where to place shell aliases or functions, or modify
# The system defaults.

# uncomment the following line to have "ll" act like "ls -l"
# alias ll "ls -l"

alias more less
alias ls "ls -F"
```

If you're running Xming (or are using a computer with any X server), you might like to use the alias `alias ls 'ls -F --color=auto'` for `ls`, which lists different types of files and directories in different colors.

Note: if you make any changes to this file you must either execute the command “`source ~/.aliases`” before they will take effect, or just log out and log back in.

9 Using the online reference information

There is an online reference manual that is referenced using the “`man`” command. For instance, to access the manual's information for `cp`, type

```
baby:~: man cp
```

If there is more than one screenful of information on a subject, advance to the next screen by hitting the space bar. If you don't want to see any more information on the subject simply type 'q' (for quit).

10 Advanced Emacs features

If you have no experience with Emacs you will probably want to skip this section for now. The introduction to Emacs in the section above is sufficient for you to be able to create and edit small files. In a couple of weeks however, after you've used Emacs a bit, be sure you return and read this section. Some tasks can be performed far more easily and quickly if you know and can use several more advanced commands, rather than having to do things the simplest way.

10.1 Line numbers and undo

goto-line	F10 e.g., or using the mouse
undo	C-_

Compilers commonly print error messages along with the number of the line in the program that contained the error. As programs become larger, it's convenient to be able to go directly to a particular line given its number, rather than searching around for it. This is done in one of two ways, depending upon whether Emacs is running in a separate window, or not (such as if you ran Emacs with the `-nw` option).

If Emacs is running in its own window you can use the mouse, and you'll see a menu at the top of the screen. Click on "Edit", move the mouse down to "Goto", and move the mouse to the right to select and click on "Goto Line...". The cursor will move down to the bottom line of the screen, and Emacs will ask you to enter a line number after the prompt "Goto line:" is displayed. If you type the number of a line in the file being edited, Emacs will jump to that line.

Even if Emacs is not running in its own window you can still use the menu commands, by pressing F10. The cursor will move to the bottom of the screen and display the first menu item. You can move through menu items by pressing the up and down arrow keys, and select them by pressing return, but you can also just type the first letter of the desired menu item. So by pressing F10, 'e', 'g', and return, you can then enter the number of the line you'd like to move to.

Sometimes you may type a command by mistake and delete or change part of your file. Emacs' undo command is very convenient, as it allows you to keep undoing the preceding commands. If you press C-_ (you have to hold down the CONTROL key while pressing SHIFT and the key with the underscore) once, Emacs will undo the last command you performed. Press it again (before performing any other command) and Emacs will undo the command before that, etc.

10.2 Commands involved in moving text

Many Emacs commands operate on all the characters within a region you specify. The region consists of all the text between something called the mark and something called the point. The point is simply the current location of the cursor. The mark is an invisible text marker in the file you place to indicate which text some command should apply to. To cut or move a block of text, you must place a mark at one end of the text and then place the cursor at the other end of the text to establish the point before indicating whether you want to copy or kill the text. In Emacs, deleting (or cutting) a block of text from a file is called killing it, and the area where the text is stored after being removed is called the kill ring. Text is moved by deleting it, moving the cursor to another location, and inserting it ("yanking

it back”, in Emacs terminology) from the kill ring. Text can be copied by placing it in the kill ring without deleting it, moving the cursor, and inserting it from the kill ring.

set mark at current cursor position	C-@ or C-SPACE (the space bar)
kill region (cut)	C-w
copy region to kill ring without deleting	M-w
yank back last region killed (paste)	C-y

To move some text from one place in a file to another, place the cursor at the beginning of the text you want to move and type C-@ or C-SPACE to put the mark here (C-@ is typed by holding CONTROL and pressing both SHIFT and @, which appears above the '2' key). Then move the cursor to the end of the text you want to move and type C-w to kill the region. Lastly, move the cursor to the location you want to move the text to and type C-y.

10.3 Multiple windows

open new window	C-x 2
open or read a different file into the current window	C-x C-f
move to other window	C-x o
switch to a different buffer	C-x b
make current window the only window	C-x 1

One advantage of Emacs is that you can display and edit more than one file at once, where each appears in a different window (the Emacs window is divided into two or more windows appearing one below the other). For instance, you could view a program's input file in one window while writing the program that is supposed to process it in another, and edit either one of them. To divide the single Emacs window in two type C-x 2. This opens a second window, where both windows are editing the same file. This may be useful if you are editing a large file and want to see one part while working on another part. To edit a different file in one of the windows type C-x C-f in one window and enter the name of the file to be edited, and that file will be loaded into that window. To move from one window to another type C-x followed by the 'o' key, and to make the window that the cursor is positioned in be the only window just type C-x 1. Or if you want to have just one window, and switch between files in it, use C-x b, then type the name of the buffer you want to switch to (each file is being edited in a different Emacs "buffer") in the status line at the bottom. (If you're connecting to the system using an X server, the "Buffers" menu at the top can be used to easily switch between buffers.)

10.4 Searching for and replacing text

enter incremental search mode	C-s	
search for next occurrence		C-s
remove last letter of search word or phrase		backspace or delete
exit search and return to starting position		C-g
exit search, leaving cursor at current position		return or enter
query replace	M-%	
y		do replacement
n		don't do replacement
C-g		exit replace mode

In editing a large program you may want to search for a particular word or phrase in the text, without having to move around the document searching for it. Typing C-s will begin an incremental search for whatever characters you type. Emacs will print “I-search:” in the bottom line, and as you type characters they will appear in this line, and the cursor will move through the text to the first point where a word or phrase containing those characters appears. For example, move the cursor to the beginning of your sample text, press C-s, and type “the”. When you type the first letter ‘t’, the cursor will move to the beginning of the word “typing” on the first line, which is where the first ‘t’ in the text appears. When you type the ‘h’, the cursor jumps to the word “this”, or the first incidence of the letters “th” in your text, and when you type the letter ‘e’ the cursor jumps to the first occurrence of the word “the” later in the same line. If you press C-s again now, the cursor will jump to the next occurrence of the word “the”; try it four times and see. Note that both “the” and “The” are found, irrespective of capitalization. If you press C-s a fifth time a beep will indicate that the word “the” isn’t found again in your text.

In incremental search mode, typing backspace or delete removes the last character of the word being searched for, and the cursor will jump backwards to the first occurrence of the new, shorter word. Pressing return or enter will exit the search mode, leaving the cursor positioned at the text that was found. Typing C-g will quit the search mode but return the cursor to the starting position.

Often it’s necessary to systematically change all (or many) occurrences of some character string to another string. For instance, you may need to rename a variable in your program. Finding and changing each occurrence manually is tedious if the variable appears many times, and it’s easily possible to miss one or more locations. Emacs allows any word or phrase to be systematically replaced with another. Typing M-% begins query replace mode, where Emacs will ask you (in the bottom line) “Query replace:” and wait for you to type a word or phrase and press return or enter. Emacs then waits for you to type a word or phrase that it will replace the first word or phrase with. Emacs will then go through the document, from the current cursor position to the end, asking at each occurrence of the first word or phrase if you want to replace it with the second word or phrase. Press the space bar to perform the replacement, or the ‘n’ key to prevent the replacement, of each occurrence of the word or phrase to be changed. Pressing C-g exits query replace mode (but doesn’t undo any replacements performed). As an example, try typing M-% followed by “tell” and “explain to”, to replace both occurrences of the word “tell” in your document by “explain to”.

10.5 Getting help

enter help system	C-x C-h	
command-apropos (while in help)		a
describe-bindings (while in help)		b
describe-key-briefly (while in help)		k

To start Emacs’ help system, press C-x C-h. The help system itself has many options and modes, of which the three most useful to you are described here. To execute these help commands, press the letter indicated when the main help system window appears:

command-apropos:	a
------------------	---

Command-apropos will display information about all commands relevant to any word you enter. For instance, if you type C-x C-h (to enter the help system) followed by the character a, Emacs will prompt you at the bottom of the screen to type in a word. As an example, if you type the word “save” and press enter or return, you will see a window listing all the commands whose names contain the word “save”. You may have to move into that window (C-o) and move down (C-v to page down)

if there are more lines than can be displayed in the window at once. If you want to run an command (perhaps you want to save your file) but can't remember the keystroke that invokes it, you may find it using command-apropos. Beware- you'll also see a number of commands that aren't discussed in this tutorial.

describe-bindings: b

Describe-bindings will list all the commands that can be used in Emacs' current mode, with the keystrokes used to execute each command. If you forget which key does what, you can run describe-bindings (C-x C-h to enter the help system, followed by b) and look for the name of the command you want to run. Beware- you'll also see a number of commands that aren't discussed in this tutorial.

describe-key-briefly: c

Describe-key-briefly will allow you to type in any keystroke, and will display in the bottom or status line the name of the command that will be executed by pressing those keys. For instance, if you type C-x C-h (to enter the help system) followed by the character c to select the describe-key-briefly mode, followed by the keystroke C-e, you'll see Emacs tells you that C-e runs the command end-of-line.

10.6 Table of all Emacs commands

C- means hold down the CONTROL key while pressing the next key mentioned.

M- means you should press the ESCAPE key, release it, and then press the next key mentioned.

Cancel command:

C-g

Basic cursor movement commands:

forward one character	right arrow or C-f
backward one character	left arrow or C-b
previous (up) one line	up arrow or C-p
next (down) one line	down arrow or C-n
one whole page up	page-up or M-v
one whole page down	page-down or C-v
beginning of current line	C-a
end of current line	C-e
beginning of file	M-<
end of file	M->

Basic commands for deleting text:

delete character forward	C-d
delete character backward	the backspace key on the keyboard
delete characters to end of line	C-k

Saving and quitting:

save file	C-x C-s
quit Emacs	C-x C-c

Line numbers and undo

goto-line	F10 e g, or using the mouse
undo	C-_

Commands involved in moving text

set mark at current cursor position	C-@ or C-SPACE (the space bar)
kill region (cut)	C-w
copy region to kill ring without deleting	M-w
yank back last region killed (paste)	C-y

Multiple windows

open new window	C-x 2
open or read a different file into the current window	C-x C-f
move to other window	C-x o
switch to a different buffer	C-x b
make current window the only window	C-x 1

Searching for and replacing text:

enter incremental search mode	C-s	
search for next occurrence		C-s
remove last letter of search word or phrase		backspace or delete
exit search and return to starting position		C-g
exit search, leaving cursor at current position		return or enter
query replace	M-%	
y		do replacement
n		don't do replacement
C-g		exit replace mode

Getting help:

enter help system	C-x C-h	
command-apropos (while in help)		a
describe-bindings (while in help)		b
describe-key-briefly (while in help)		k

10.7 Conclusion

Emacs has many more capabilities than those mentioned here. This tutorial has discussed only the basics necessary for you to be able to create and edit programs as necessary for your course, plus a very few useful additional features. If you want to learn more about Emacs than was presented above try any of the following sources of information:

1. The Help Desk in room 1400 of the Computer and Space Sciences building has information on Emacs. An online version is available via the following link:
www.helpdesk.umd.edu/documentation/unix/emacs.shtml
2. An online Emacs tutorial is available. To see the tutorial, run emacs (type emacs at the UNIX prompt), and when it starts press the ESCAPE key, release it, hit the x, type help-with-tutorial, and press return (or enter). The words “help-with-tutorial” will appear in the bottom line of the Emacs screen while you are typing them. Follow the tutorial instructions after that.
3. The libraries and bookstores have books on Emacs, as well as books on UNIX that have sections or chapters about Emacs, if you think you might need to continue using UNIX and Emacs in the future.