

A UNIX TUTORIAL

Jandelyn Plane
Larry Herman
Charles Lin
Jeff Hollingsworth

Table of Contents

Introduction.....	2	Chart of all Emacs commands.....	10
Getting Started	2	Conclusion	12
Logging in to your class account	2	Printing A File.....	12
Changing your password.....	3	Directories	12
Logging out.....	3	Introduction.....	12
Registering the Account for Identification	4	Listing files in current directory.....	13
The EMACS Text Editor	4	Creating a new directory	13
Introduction.....	4	Changing directories	14
Cancel command.....	5	Listing files	14
Creating a file.....	5	Creating new files in the current directory...	14
Editing a file.....	6	Copying a file.....	14
Deleting characters.....	6	Renaming (moving) a file	15
Deleting lines	6	Removing (deleting) a file	15
Adding a line.....	6	Directory hierarchy	15
Replacing characters	7	Removing (deleting) a subdirectory.....	17
Saving a file	7	Wildcard characters.....	17
Quitting Emacs.....	7	Disk quota	17
Opening an existing file.....	7	Compiling And Running C Programs	18
Chart of basic Emacs commands	7	Compiling a program	18
Advanced Emacs features.....	8	Other options to the gcc command.....	18
Line numbers and undo.....	8	Executing a program	19
Commands involved in moving text.....	8	Stopping a running program	20
Multiple windows	9	Job Control.....	20
Searching for and replacing text	9	Submitting Programs.....	20
Getting help.....	10	Using The On-line Reference Manual	20

Introduction

This is a brief introduction to the UNIX operating system and the details of the use of the command-line interface for the CMSC 212 course. This tutorial covers procedures for logging into the system, manipulating files and directories, compiling and running programs, submitting programs electronically, and other information. This tutorial is only an introduction, and not a comprehensive study of UNIX. UNIX is very powerful, and we will only discuss the most important capabilities, and those that you need for doing your work in this course.

Information which is to be typed exactly as it appears usually follows a percent symbol (%) which is the default T-shell UNIX prompt, or is indicated in the discussion above the command. Angle brackets (< >) enclose information you should fill in such as <filename> will be replaced with the actual name of a file. Explanations will appear in regular type. Also, you should be aware that UNIX is case-sensitive, that is, it distinguishes between lowercase and uppercase letters, and so you will need to type commands exactly as described. The UNIX command-line uses "white space" as a delimiter. This means that the parts of the command line are separated by anything that appears as blanks space (pressing the space bar, pressing the tab key or even a end-of line, when it is done right). The problem with this is that you may be used to file names containing spaces as they do in Windows, this causes problems in UNIX unless you surround the file name in quotes.

For the CMSC 212 class you will be using the Linux WAM machines as your development space and the Linux machines in CSIC as the CVS server. Both of these are actually Linux machines. Linux is a version of UNIX. This tutorial mostly introduces you to UNIX shell command-line in general, but there are things specific to the class which will be noted at that time.

To use this tutorial, you will first need to have three pieces of information:

- 1) Your class account ID, denoted in this exercise as <classID>.
- 2) The password for that account, denoted as <classpassword>.
- 3) Your account on the CVS server, denoted in this exercise as <CVSID>.
- 4) The password for that account, denoted as <CVSpasword>.

Getting Started

Logging in to your class account

There are three steps involved in logging into a system: specifying the machine you want to use, giving your login id, and giving the password for that login id.

For this class you will be using WAM Linux machines for your project development space. To start practicing the use of the UNIX command line, you should first be on one of those machines.

The easiest way to get onto one of those machines is to go into one of the Linux WAM labs around campus. One of those labs is in A.V.Williams room 1120. Most of the machines in that room are Linux based PCs. There are 4-5 Windows NT machines in that room (they are tan in color). There are also 3 Windows 2000 machines in that room (they are black in color like the Linux machines). When you sit down in front of any of those machines it should ask for your login id and password. You will give your class account id and class account password in this screen. It will then open windows in which you can do your class project development.

If you are not in the lab, you will be able to use a secure shell application in order to do your assignments from another location. The specific details of this are still being worked out.

When you first log on, if any news articles about the Cluster systems are unread then the news program may automatically be run so you can read them. The prompt telling you that you are reading news is [ynq] by default, and to quit out of news press q (a prompt is a symbol used to indicate that the system is ready for input; it will change as you go from the news system to the mail system to UNIX). If you have any electronic mail then the command line mail program will automatically be run after that. The prompt for the mail program is &, and you

can type x to quit out of mail. You should then see the default UNIX system T-shell prompt, which is a percent sign (%).

The UNIX commands given below are the same in all of the shells - there is only a difference at the shell programming level and at the level of short-cuts available for the commands. This means that as long as these commands are given at a shell prompt, they will be given and act the same so it does not matter which shell you are given.

Changing your password

The password for an account is the major part of the security for that account. In order to ensure that no one else has access to your account, your first action upon receiving a new account should be to change the password to something that you will find easy to remember, but only you will know.

There have been some problems over the last few years with people breaking into other people's accounts by figuring out passwords. If your account is broken into then all your files could be destroyed. To minimize the chances of such an occurrence, use the following guidelines in choosing a password for your account.

- a) Do not use any word from the dictionary or anyone's name.
- b) Do not use any part of your name, your login name, or any other name which someone who knows you might know to try.
- c) A password for the UNIX system should be eight or fewer characters, six to eight preferably. It should include a mix of upper and lower case letters, digits, and punctuation. Recent guidelines say that digit(s) should not be the first or last character(s) of the password.
- d) Do not use any pattern on the keyboard.

Never email your password, and do not tell your password to anyone!

To change your password, at the % prompt simply type:

```
% passwd
```

You will be prompted for the necessary information to change your password. Answer all the questions. (The request for you to "verify" your new password is simply a request to type the new password again.) When you are done, your new password will be the password for your account until you change it.

On some systems there is additional software running to make UNIX more secure. One example of this is the CSIC Linux Cluster which is running Kerberos. Because this extra software must be consulted during things like password changes, the command to change the password is from Kerberos rather than the UNIX shell directly. The command is

```
% kpasswd
```

but works exactly the same as the passwd command will run on the WAM systems.

Logging out

After learning how to log in and change the password, you should next learn, before anything else, how to log out. For an orderly exit, at the prompt type

```
% logout
```

You could also type

```
% exit
```

to close the login session if you prefer. If you are in an environment with many login session windows open, you should logout or exit from each of them to make sure there was nothing running that should have been terminated.

Closing the secure shell window is not an appropriate way to close the login session. You should always direct UNIX to close the session before closing the connection from a remote machine.

When working in a multi-window environment, such as the X-Windows like setup on the Linux machines in the WAM lab, you must close each window individually and the environment separately if it doesn't close on its own. Before you walk away from the computer, always make sure the screen shown is the one that is requesting the next users password. To close all of them you would type exit or logout in each window - when the lowest level accessed in that window is exited, the window will close. If the desktop environment is still displayed, use the pull-down menus present at the top of the screen to close the session.

Registering the Account for Identification

The account has associated with it identity information which should tell other people on the system the name and other data about the owner of that account. When the class accounts given to students, this information is about the class rather than about the student who has that account. To change the information so that it tells your name, you can type the chfn command at the % prompt:

```
% chfn
```

This utility will then ask a list of questions. We require that you put your real name (as you are registered for the class) into this registration. The other information is your option - if you enter your phone number, please realize that anyone logged into wam can see it (it is like having a listed number rather than unlisted). For each question asked, the default answer is shown in square brackets.

To see this information about yourself (or any other user), use the command finger (often abbreviated to f) at the % prompt followed by any login id. For example if you want to find out the information about the user with the login id jplane type the following:

```
% finger jplane
```

This would display the real name, office number and phone number about that one user.

The EMACS Text Editor

If in the past you have used integrated development environments such as Eclipse to write programs, you will find the development process in UNIX is a little different. Although a number of more sophisticated program development tools are available in UNIX these are beyond the scope of your course, therefore the simplest program development utilities are described here.

To write and run a C program in UNIX you would first use a UNIX 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 exit the text editor and run the C compiler, which is a separate program. The compiler produces an executable program as a file in your account, 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 any step (the program fails to compile, or it compiles and runs but doesn't work correctly) you would have to run the text editor again to edit the program, figure out what's wrong and make changes to it, and save it again so you can exit and compile again.

Some students may know the Pico text editor, available in most UNIX installations. A simple text editor like Pico is suitable for editing very small files or email messages, but is inadequate for creating larger programs such as you will write in this course. If the only text editor you know is Pico this is the time to learn and begin using a better one. The two most common general-purpose UNIX text editors are vi and Emacs. If you already know vi feel free to use it for your projects. This tutorial explains the basic Emacs commands. The best way to learn these commands is to experiment and practice with them.

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). C-f indicates that you should

press the CONTROL and, while holding the CONTROL key, then press the 'f' key. On the other hand, M-v means you should press ESCAPE and release it, and then press 'v'.

Cancel command

Before you start Emacs, you should know about 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.

Creating a file

In the following exercise, you will invoke Emacs, type some text, and save that text in a file called funfile.

```
% emacs funfile
```

If you are logging in from a Sun workstation and Emacs doesn't start, and an error message about a DISPLAY environment variable is printed instead, run Emacs with the "-nw" option (e.g., "emacs -nw funfile").

When working in a multi-window environment, such as the X-Window like setup on the Linux machines in the WAM lab, emacs will open a new window automatically so you do not need to use the -nw option. The benefit of having it open in the new window is that the old window can still be used for giving commands such as compiling the file as long as you still have access to a shell prompt. In order to have it start in another window in such a way that you have access to a shell prompt, type a & at the end of the command line - immediately before you press the enter key. This tells the process to run in the background. (See the section about controlling jobs for more on this topic.) When the command

```
% emacs filename &
```

is given, a prompt will be displayed on the very next line and Emacs will open in another window. When compiling in one window while editing that same file in another window it is important to remember the distinction between the file and the buffer. Remember the commands for compilation given at the UNIX shell prompt will only compile files - they will not compile something that is in the buffer. When you are editing in Emacs, you are always editing a buffer, a temporary copy of the file, you are never actually editing a file. You will write the contents of the buffer to a file when you tell Emacs to save the buffer. The important point here, is before you try to compile the newest version of what you see in your Emacs window, be sure you have saved the buffer to a file.

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 which Emacs may ask you. Type the following sentence fragment:

```
I am typing this files in order
```

Note that when you first typed a character, "***" appeared 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, in this case, it has never been saved). Now before actually entering any more text into your file, try saving your work and exiting Emacs. Type C-x C-s to save (hold down the CONTROL key and press x, then hold down the CONTROL key and press s). This saves your file. To exit Emacs type C-x C-c, and you should find yourself with the UNIX prompt. Now reenter "funfile" with the command:

```
% emacs funfile
```

One other command is very useful to know before going further with this exercise. If at any point you want to cancel a command, type C-g.

You are now ready to enter some more text. By default, Emacs does not have the word wrap feature 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. Don't worry if you make typing mistakes, you can correct them later as a part of this exercise. To get practice with Emacs you could type a dry paragraph from any book, but it will be more fun to type a poem written by Shel Silverstein. Type the following:

```
I am typing this files in order to to test the features of the UNIX  
text editor Emacs.
```

```
This board that we just built is just fine--
And don't try to tell us it's not.
And don't try to tell us it's not.
It's the bottom I guess we forgot....
```

Editing a file

The information below presents the basic commands for modifying or editing files. Basic cursor movement commands

forward one character	C-f
backward one character	C-b
previous (up) one line	C-p
next (down) one line	C-n
beginning of current line	C-a
end of current line	C-e
one whole page up	C-v
one whole page down	M-v
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. Note that although you've learned the Emacs way of moving the cursor forward and backward, up and down, we should say that usually the arrow keys will work when you are in Emacs.

Basic commands for deleting text

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

Deleting characters

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. 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).

In the first line of text you've typed the word "files" is plural when it should be singular. Move the cursor to the blank after the "s" in "files" and press the backspace key. Later in the line the word "to" appears twice. Delete one incidence of "to" by moving the cursor to the "t" of one of them. Type C-d three times to delete "to".

Deleting lines

The second line of the poem was repeated unintentionally. To delete the second line of the poem, move the cursor to the second line and type C-k. The text should be gone, but a blank line remains. Type C-d, or C-k again, to remove the end-of-line marker and remove this blank line.

C-k will remove or delete all of the current line after the current cursor position, so if you want to save the beginning of a line but delete the last part you can use it to do so.

Adding a line

The third line of the poem is missing. To add a line below the second line, move the cursor to the end of the second line and hit RETURN to open up a new line. Alternatively, you can move the cursor to the beginning of the third line and type C-o to open up a new line after the current cursor position. In the space which has opened up for the third line type the following:

```
The sides and the back are divine--
```

Replacing characters

In the first line of the poem there is a typographical error which affects the meaning of the poem drastically. The word "boar" must be changed to "boat." A couple of words later on this same line "ahtt" must be changed to "that." Use the commands you have learned above to accomplish these changes.

Saving a file

So far the work you have been doing 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, funfile, 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 the 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 save at least one backup copy in a different directory or with a different filename, using the UNIX commands explained below.

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) 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.

Opening an existing file

If you have exited funfile, to return to that file for modification, type:

```
% emacs funfile
```

All of the new window information given in the section about opening Emacs to create a file also applies to opening Emacs to edit a file that already exists.

When you edit an existing file and save any changes to it, Emacs will save the original version of the file as a backup copy by renaming it. The backup copy will have the same name as the original file but will have a ~ character at the end. When you quit Emacs and list the contents of your directory, you should see the backup file there. 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 status you can use the contents of this backup file, after renaming it (if you need it, you should rename the backup file to prevent Emacs from overwriting it when you next edit the original file).

Chart of basic Emacs commands

If you're a new user you can print this chart 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	C-f
backward one character	C-b
previous (up) one line	C-p
next (down) one line	C-n
beginning of current line	C-a
end of current line	C-e
one whole page up	C-v
one whole page down	M-v

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, or C-h, which also does this
delete characters to end of line	C-k

Saving and quitting

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

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.

Line numbers and undo

goto-line	C-c g
undo	C-_

Compilers commonly print error messages along with the number of the line in the program which 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. If you press C-c followed by g, Emacs will ask you to enter a line number in the bottom line of the screen. When you type in a line number and press return, the cursor will jump to that line.

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.

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 the explanatory text from the beginning of funfile to the end of the file, place the cursor at the beginning of the first line of the text (the first character of the file). 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). Now move the cursor to the end of the sentence to establish the point. Type C-w to kill the region, and move the cursor to the second blank line following the poem. Type C-y and the paragraph should now follow the poem.

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
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 which 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 which the cursor is positioned in be the only window just type 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

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 which 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 which 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".

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 a command (perhaps you want to save your file) but can't remember the keystroke which invokes it, you may find it using command-apropos. Beware- you'll also see a number of commands which aren't discussed in this tutorial.

describe-bindings: b

Describe-bindings will list all the commands which 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 which 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 which 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.

Chart 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	C-f
backward one character	C-b
previous (up) one line	C-p
next (down) one line	C-n
beginning of current line	C-a
end of current line	C-e
one whole page up	C-v

one whole page down	M-v
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, or C-h, which also does this
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	C-c g
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
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

Conclusion

Emacs has many, 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 a free handout on Emacs. An online version is available at <http://www.inform.umd.edu/CompRes/Docs/UNIXSupport/emacs.html>
- 2) An on-line 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 which have sections or chapters about Emacs.

Printing A File

From your WAM account can use the laser printers in WAM labs, but you must first get a print account and pay by the page. Having a print account can be convenient, so you don't have to walk to a different building to get a printout. Information on getting a print account and using pay-for-print is on OIT's webpage; see the link from the class webpage.

The command to print one or more files to the OIT Postscript printers is:

```
% qpr -q csc <names-of-files-to-print>
```

If you want to save paper you can use the "-x duplex" option, which prints on both sides of the paper:

```
% qpr -q csc -x duplex <names-of-files-to-print>
```

If you want to really 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:

```
% mpage -2Hftm30lr -Pcsc <names-of-files-to-print>
```

Note there is no space between "-P" and "csc". 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. The meaning of each option in "-2Hftm30l" isn't really worth explaining, but since this command is rather cryptic you might really prefer to create an alias for it; see alias section below.

Directories

Introduction

Directories are where files are stored, 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 which 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. When you first log in, your current directory is your own account's home directory. You can only change to directories for which you have been given permission.

In UNIX a single period (.) refers to the current directory, and two periods (..) refers to the directory one level above (closer to the root) the current directory; this is called the parent 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 the login id you were given in class and will not change through the entire term; you will be in your home as you logon each time. The current directory corresponds to your home directory as you first logon, 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 most commands.

A pathname is a description of where a file (usually one which is not in the current directory) appears, including the directory names which form a path to the directory where the file is located. The slash (/) designates the root directory, but it is also used to separate directory names in a pathname. For instance, the pathname /usr/skel/default.login refers to a file named "default.login" located in the "skel" subdirectory of the "usr" subdirectory of the root directory. Files which are not in the current directory are sometimes specified using a pathname showing where they are in relation to the root directory, as in /usr/skel/default.login. They are even more commonly specified relative to the current directory or relative to a home directory (yours or another user's), using ~. The examples below show different ways of referring to files and use the symbols mentioned above.

Listing files in current directory

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

```
% 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 two which 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 files type

```
% ls -a
```

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

```
% ls -l
```

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

```
-rw-r--r--  1  jplane          student          2565  Nov 16 1995  office
drwxr-xr-x  2  jplane          student           512   Jul  5 1996  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.
- 3) The third column specifies the name of the owner of the file, which will be your <loginID> for the files you create.
- 4) The fourth column specifies the name of the group the file is associated with. This can be changed with the chgrp command.
- 5) The fifth column has the size of the file, in bytes.
- 6) The next columns have the date of creation or of last modification of the file or directory.
- 7) The last column gives the name of the file or directory.

If you wish to determine the type of the things listed in the directory, but do not want the long listing produced by ls -l, you can type:

```
% ls -F
```

This will list the contents of the current directory in something similar to the following:

```
office  prereg/  a.out*  this@
```

Directories will end in a /. Files that can be executed will end in a *. Names that are actually links to something stored in another directory are marked with a @ at the end. Normal text files usually don't end in anything special. In the example, office is a normal file, prereg is a directory, a.out is an executable, and this is a symbolic link. Even though there is a /, a * or an @, the name of the directory is still prereg (not prereg/) and the name of the executable is still a.out (not a.out*).

Creating a new directory

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

```
% mkdir literature
List your current directory to see that this new directory has been created.

% ls
% ls -l
```

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 similar to how you can move from room to room within a building. Change to this new directory:

```
% cd literature
```

Listing files

If you just created the literature directory and you are going in before you have put any children into that directory, you will see that there are no non-hidden files in this directory at this point.

The `ls` command without any options or arguments will give you nothing to list. If you use the command that includes the `a` option, you will see the hidden children of this directory named `.` (current) and `..` (parent) which help the directory keep track of itself and its own location in the tree.

```
% ls -a
```

The `ls` command can also be given with an argument. If no argument is given, the command assumes you want to list the children of the current working directory. If an argument is given and that argument corresponds to the name of a directory, the `ls` command will list the contents of the directory named rather than the contents of the current working directory.

While you are located in the literature subdirectory, the files in the parent directory can be listed by using the `..` as the argument to the `ls` command,

```
% ls ..
```

or the files in your own home can be listed using the `~` as the argument to the `ls` command,

```
% ls ~
```

Since the literature directory was created as a child of your own home, these two commands would do exactly the same thing.

Creating new files in the current directory

Using a text editor, 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.

Copying a file

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

```
% 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. Since it is an ASCII file, the `more` command can be used to display the contents of that file.

```
% more file3
```

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.

The first of the two arguments to the `cp` command will always tell "from where" it should be copied and the second will always tell "to where". If the first argument is a filename, the second can be either a directory or a

file. If it is just a filename, a new file with that name comes into existence or it overwrites a file that already exists. If it is the name of a directory. A new copy of the file is created in the directory named, but the filename is the same as the first argument.

The original file1 is in the literature directory which is a child of your home. To make a copy of file1 in your home directory with the same name you can use any one of the following:

```
% cp file1 ..
% cp file1 ~
```

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

```
% cp file1 ../testfile
% cp file1 ~/testfile
```

The first one of these 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 lh123 and you knew that it was located, for example, in /tmp_mnt/home/cmssc106lh, 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 /tmp_mnt/home/cmssc106lh/testfile", but since the full path name is extra work to type there isn't much advantage to doing so.

Renaming (moving) a file

The command to rename a file (or move its logical location) is mv. It also takes two arguments, a "from where" as the first argument and a "to where" as the second argument. To move file1 to file4.

```
% mv file1 file4
```

Look at file4 to see what it has in it.

```
% more 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 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.

Removing (deleting) a file

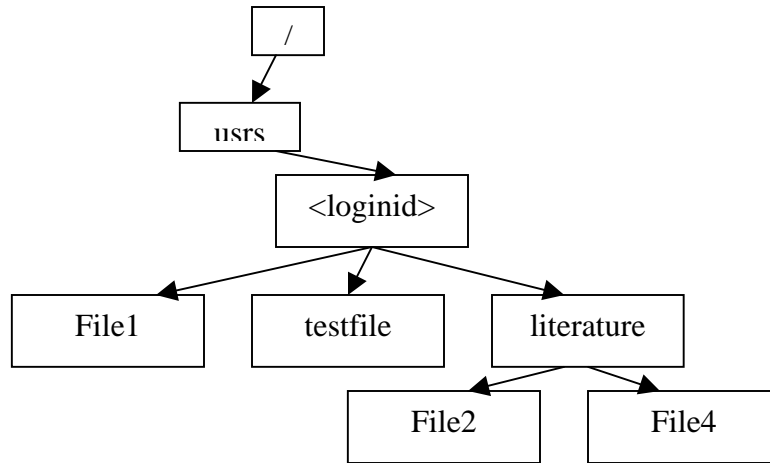
The command to delete a file is rm. This command can take one or more arguments. The arguments are the names of the files to be deleted. Realize deleting a file is more permanent than the equivalent "moving it to the recycle bin" of the windows environment. Once something is removed - the only way to get it back would be the version that existed when the last backup was done. Depending on the system, it may not be possible at all. An example of the command is:

```
% rm file3
```

If you list the contents of the directory after this command, you will see that file3 no longer exists.

Directory hierarchy

A diagram of a hierarchical structure could look like this.



Notice your "account" is just a subdirectory in the directory of accounts for the class. 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:

```
% pwd
```

The first slash in this listing of directories represents the root. The names which follow represent directories leading directly from the root to your account directory. The slashes other than the first are delimiters.

To change to the next higher directory (the parent directory), use the command:

```
% cd ..
```

To change back to your own home directory at any point in time, use the command

```
% cd
```

Notice the command `cd` without any argument moves you to your own home. The default value for the `cd` command is the home directory - it is one of the few where the default is not the current directory. (If the current directory was the default, it would say to move me from the current to the current which would be useless.)

To move into any directory which is a child of the current working directory, you would just give the name of that directory:

```
% cd childdir
```

To move to any directory that is not related to the current directory in the hierarchical structure, you can fully name it:

```
% cd /this/that/other
```

This example would change to the directory named `other` which is a child of the directory named `that` which is a child of the directory named `this` which is a child of the root.

To summarize there are four ways to name a directory or a file.

Beginning Character	Meaning
/	you start at the root of the structure and work from that point
..	you start at the parent of the current directory and work from that point
~	you start from a home (your own home if it is ~/ or from the home indicated by the login id if it is ~id.)
Anything else	you start from your current directory.

For example:

<code>~this/that</code>	means that which is a child of the home of the person with the login id this
<code>this/that</code>	means that which is a child of this which is a child of the current working directory
<code>../this/that</code>	means that which is a child of this which is a child of the parent of the current working directory (in other words, this is a sibling of the current working directory)
<code>/this/that</code>	means that which is a child of this which is a child of the root

Removing (deleting) a subdirectory

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

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.c
answer3.txt  answer6.txt  hw2          hw3.ans     proj1.c      proj4.c
answer4.txt  hw1          hw2.ans     program1     proj2.c      txt
```

The command `"ls hw?"` lists all files whose names are "hw" followed by any other character, in other words, 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?.c"` lists the files `proj1.c`, `proj2.c`, `proj3.c`, and `proj4.c`.

The command `"ls pro*"` lists `program1`, `program2`, `proj1.c`, `proj2.c`, `proj3.c`, and `proj4.c`.

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

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

```
cp ~fred/*.c .          copy all files whose names end in ".c"
                        from the user fred's directory to your
                        own
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.

Disk quota

Every account has a limit to the amount of information which can be stored. To see what your limit is and how much space you have available, type the command `"fs lq"`. 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. Be careful not to remove any important files, because once you delete a file it cannot be recovered.

One common way for students to go over their disk quota during a semester is to have one or more files named "core" in their directories. The core file is created whenever an application ends abnormally. It was originally intended to help the programmer figure out what went wrong to cause the program to end abnormally. It is often a very large file which is why it causes you to quickly reach your quota. It is called core because it is a picture of what "the core" memory looked like at the exact moment the program crashed. One way to debug programs is to read the core file to know exactly what state things were in when the program crashed, but programmers seldom use the core file directly like this anymore so the best solution for you is to delete that core file using the rm command.

Compiling And Running C Programs

Compiling 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 one C compiler is "gcc", and if you had a C program in a file named test.c it could be compiled like so:

```
% gcc .c
```

If your program had any syntax errors you would get error messages. Using Emacs, or another text editor, you would have to correct these errors and recompile. Sometimes you will have more than a single screenful of error messages. It is very difficult to read error messages which fly by so quickly, 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:

```
% gcc .c >& myerrors
```

The >& tells the operating system that the results of the command to the left (which compiles the program) are not to be printed to the screen, but are instead to be written to the file name which 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.

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

```
% ls -l
```

You will notice that there is executable permission on a.out since it is an executable file. If you use the -o option you can tell the compiler to put the executable program in a file with a different name than a.out. For instance,

```
% gcc -o test1.c
```

puts the executable program in a file called test1. There is one major advantage to using an executable file name other than a.out. Only one file in a directory can have the name a.out at any time, so if you want to run several different programs you have to compile each one first, but if you compile them and give them different executable file names then you can run them afterwards as many times as necessary, without compiling again.

Other options to the gcc command

-g	Enable Debugging
-Wall	Warn about common errors
-c	Only compile to object code
-fprofile-arcs	Count how often statements run
-ftest-coverage	Test program coverage

These will be discussed in more detail in class.

Executing a program

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

```
% test1
```

If this program has any output to the screen, you will see that appear on the screen before the next UNIX (%) prompt. Reading and verifying this output will tell you if your program is producing the correct results. If this information is going too fast and scrolling off the screen so that you don't have a chance to read it, you can combine the execution command with the "more" command which causes that display to be one screenful at a time. The pipe "|" is used to connect two commands so that the output of one is taken as the input to the next. The following command executes the program named and displays the output one screen at a time.

```
% test1 | more
```

The information printed from a program is printed to "standard output". By default, standard output is your screen. If you would rather have this information go into a file so that you can print it or view it later instead of displaying anything to the screen, you can redirect "standard output" using "output redirection" and have the output sent into a file. If the file named already exists, its contents will be overwritten, so be careful that you are not replacing the contents of a file you want to keep. The following command will execute the file named test1 and the output will be redirected into the file named test.output while nothing appears on the screen.

```
% test1 > test.output
```

After giving this command you can display the contents of the output file to the screen with the command

```
% more test.output
```

or print the file with the command

```
% qpr -q pr1 test.output
```

because it is an ASCII file created.

If this program expects any input from the user it will wait at that point for the user to type something and then press the enter or return key. In UNIX parlance, the program reads its input from "standard input". By default, standard input comes from the keyboard (anything typed in, followed by the enter or return key). If you would like the input to come from a file instead, you can use input redirection to redirect "standard input". This will cause the input to be read from a file rather than from the keyboard. The following command will execute the file named test1 and any input it needs will come from the ASCII file named test.in.

```
% test1 < test.in
```

It will not wait for any input from the user if input redirection is used. The file test.in would need to have been created as an ASCII file most likely using an editor, and it must be in the exact correct format expected by the file test.x. As far as test.x is concerned, it receives its input from the outside world through "standard input" and it can't tell whether this input came from the keyboard or from a file. It just sees a stream of characters and processes it accordingly.

Input redirection and output redirection can both be used. This would cause test.x to read information from the file named test.in and put the output into the file named test.output.

```
% test.x < test.in > test.output
```

If more than one file is needed for input, you can use the cat command and the pipe "|" to have the files read by the executable program in sequence. The file named first will be completely read in order before the second file is started. Just like with input redirection, the files must be in the exact format required by the executable. Giving the following command will cause the file named test.1 to be read followed by the contents of test.2 both as input to test.x.

```
% cat test.1 test.2 | test.x
```

Also the output produced can be put into a file by adding output redirection to this command.

```
% cat test.1 test.2 | test.x > test.output
```

By default, the cat command prints its output to standard output. If you provide more than one file, it will print the files from left to right. In the following example, test.1 will be printed to the screen, followed by test.2, followed by test.3. cat can take zero or more files as arguments.

```
% cat test.1 test.2 test.3
```

cat does not change the original files; it just prints the contents to the screen. You may want to "glue" (technically, concatenate, or "cat" for short) two or more files together and save it to another file. For example, the following will create a new file called test.output, which is the "concatenation" or the gluing of _contents_ of test.1 followed by the contents of test.2. Neither test.1 nor test.2 is changed.

```
% cat test.1 test.2 > test.output
```

Make sure not to use the same file name on the left of the redirection operator and the right. For example, the following will cause a problem since test.1 appears in two locations.

```
% cat test.1 test.2 > test.1
```

Some versions of cat will prevent you from doing this.

Stopping a running program

If a program of yours doesn't seem to be working right you can always stop it by typing ^c (hold down the control key and press c). Most UNIX commands can be stopped as well by typing ^c.

Job Control

In UNIX, you can run several programs at once from a single command interpreter. To do this, you can start a job in the background by adding the & character at the end of the command line. For example:

```
% emacs foo.c &
```

To learn what programs you have running, you can use the jobs command. For example:

```
% jobs
```

will list all of the jobs you current have running. The listing will look something like this:

```
[1] - Suspended vi foo.c
[2] + Suspended ssh wam
```

You can terminate any of these jobs using the kill command. For example "kill %1" would terminate the first job in the above listing. It is always a good idea to run the jobs command before logging out since the logout command won't work unless you terminate all of your jobs first.

Submitting Programs

You will submit your assignments to the submit server (the same one you used in CMSC 132, but now will use the command line interface). The command line interface is in ~hollings/bin/submit on the WAM machines. To run the command, you type ~hollings/bin/submit from the directory containing your project. This command will show a listing of the files that have been stored into the submit server. You should verify that your submission is correct by going to the submit server web page at <http://submit.cs.umd.edu>. You can login to this system using your directory id and password.

Using The On-line Reference Manual

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

```
% 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).

If you do not remember the exact spelling of the command name, the man command can be used with the -k followed by a keyword instead of a command name. It will then list all of the commands that have that keyword in their description. For example if you can not remember that the command to change the password is passwd, you can use the command

```
% man -k password
```

to display the commands that deal with passwords.