Prof. Dr. rer. nat. habil. Martin O. Steinhauser
National Defense Academy, Yokosuka, Japan
Frankfurt University, Germany
October 2024

Computational Materials Science
with
Atomistic and Coarse-Grained Methods

Handout 2

Basics of the UNIX programming g environment

1 Introduction

In writing C/C++ programs to run under Unix[1] , there are several concepts and tools that turn out to be quite useful. The most obvious difference, if you are coming from a PC programming background, is that the tools are separate entities, not components in a tightly coupled environment like Metroworks CodeWarrior or Microsoft Visual C++. The most important tools in this domain are the editor, the compiler, the linker, the make utility, and the debugger. There are a variety of choices as far as "which compiler" or "which editor", but the choice is usually one of personal preference. The choice of editor, however, is sometimes almost a religious issue. Emacs[2] is one of the oldest editors available on Unix systems; it integrates well with the other tools, has a nice graphical interface, and is almost an operating system unto itself, so I do encourage its use. The appendix at the end of this handout gives a summary of some basic UNIX and Emacs commands to get you started. For beginners, the use of Emacs as an editor seems to be overly complicated due to the many customization possibilities. In fact, Emacs as an editor for writing scientific programs has by now been superseded by so-called *Integrated Development Environments* (IDEs) which are very powerful and integrate all the different programming tasks (editing, compiling, debugging, even running the program) in one single Graphical User Interface (GUI). During the course of this lecture, I will perhaps introduce you to one very widespread IDE available for free, namely Eclipse (www.eclipse.org).

1.1 Caveat

This handout is not meant to give a comprehensive treatment of any of the tools discussed, rather it should be used as an introduction to get yourself started with the tools. If you find that you need more information, all of the programs have extensive man pages and xinfo entries, as well as some on-line help for all of its commands. These man pages list numerous bits of information, some of which will appear to be quite attractive, but if you do not know or understand what an option does, especially for the compiler, please do not use it just because it sounds nice. Also, O'Reilly & Associates publishes a pretty good set of references for basic Unix tools, the titles of which resemble "UNIX in a Nutshell". These are not required or endorsed, but may be useful if you get lost.

[1] Unix is a registered trademark of AT&T.

[2] GNU Emacs is provided for free by the FREE SOFTWARE FOUNDATION, which write software and gives it away because they think that's how software should be.

2 The Editing Process – Emacs or Xemacs

The first frontier a person is confronted with when programming is the editing environment.

As mentioned, the choice of editor in UNIX was and still is, for various valid and not so valid reasons, pretty much set in stone as the 11th commandment: Whosoever writes programs with Emacsi shall live without bugs, or else... Emacs has pretty much all the features you will possible need in an editor, and then some; the only problem is finding out how to invoke them as needed. This section is only meant as a quick reference to get you started. To start editing a new or existing file using Emacs, simply type this to the UNIX prompt: *emacs filename*, where *filename* is the file to be edited. Once Emacs has started up, text entry works much the same way as in any other text editors, i.e. the default mode is INSERT mode where text being typed is inserted at the current cursor position. All the fancy editing commands, such as *find-and-replace*, are invoked through typing special key sequences (there really isn't a great mouse-driven UNIX editor for writing code). Two important key sequences to remember are: *^x*(holding down the "ctrl" key while typing 'x'), and *[esc]-x* (simply pressing the "esc" key followed by typing 'x'), both of which are used to start command sequences. Note that in most user manuals for Emacs, the "esc" key is actually referred to as the "Meta" key (lets not bother explaining that one). Therefore, you will often see commands prefaced with the key sequence *M-x*, as opposed to the *[esc]-x* that we will use in this handout. Since the two are pretty much synonymous, we'll stick with explicitly using the "esc" key. Now let's see some examples of these mysterious "command sequences".

For instance, to save the file being edited, the sequence is *^x ^s*. To exit Emacs (and be prompted to save), the sequence is *^x ^c*. To open another file within Emacs, the sequence is *^x ^f*. ('f' for "find file"). This sequence can be used to open an existing file as well as a new file. If you have multiple files open, Emacs stores them in different "buffers". To switch from one buffer to another (very handy when you are editing a .c source file and need to refer to the prototypes and definitions in the .h header file), you use the key sequence *^x* followed by typing 'b' (without holding down "ctrl" when typing 'b'). You can then enter the name of the file to switch to the corresponding buffer (a default name is provided for fast switching). The arrow keys usually work as the cursor movement keys, but in the event that they don't, *^f* is for moving forward (right), *^b* is for moving backward (left), *^p* is for moving to the previous line (up), and *^n* is for moving to the next line (down).

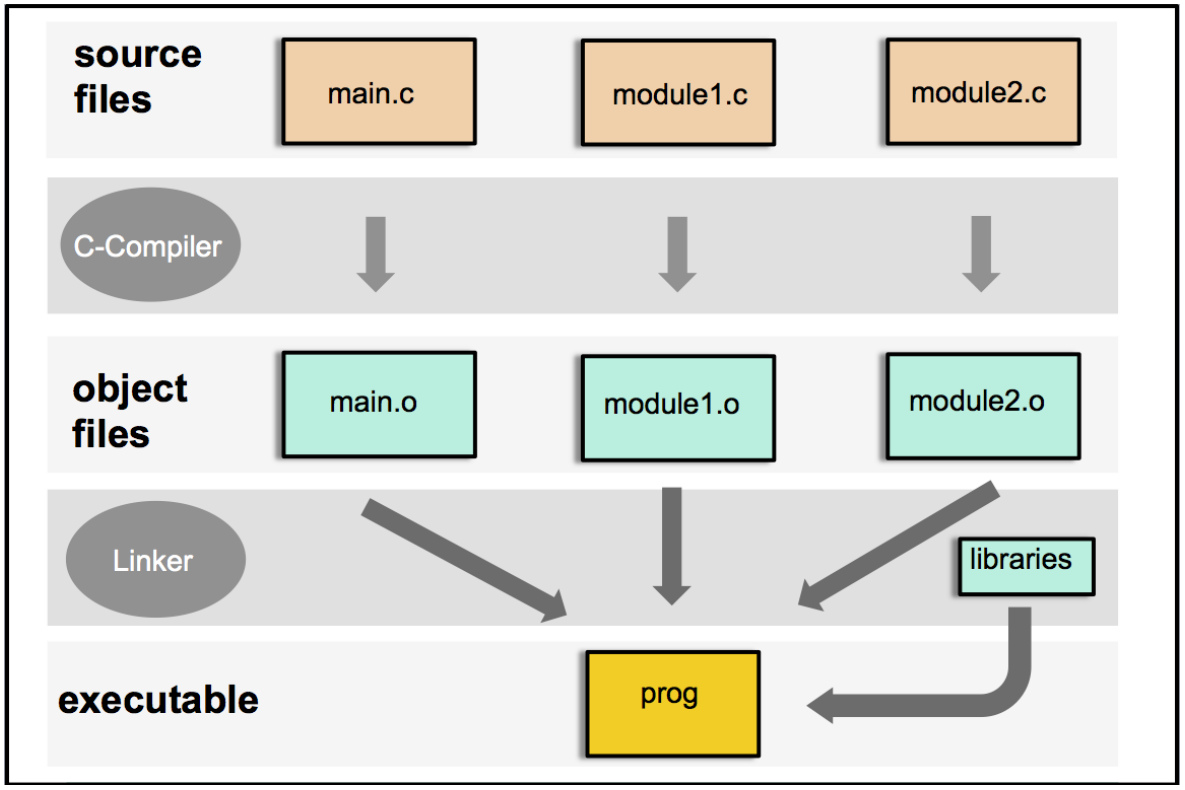
[esc]-< ([esc] followed by '<') moves to the beginning of the file, and *[esc]->* moves to the end of the file. Finally, *^v* does a "page down" and *[esc]-v* does a "page up". Note that the [esc] key is simply pressed and not held, whereas the [ctrl] key is held in all cases. Quite frequently you'll make typing mistakes. The backspace key usually deletes the character before the text cursor, but sometimes it is strangely bound to invoke help! In case that fails, try the delete key (the one on the mini-keypad that also contains page up, page down, etc). You can also delete a whole line at a time by using *^k*. Another important editing feature is copy/cut and paste. To copy or cut, you first have to select a region of text. To tell Emacs where this region begins, use *[esc]-@* or *^[space]* (control and space simultaneously). Then move the text cursor to the end of the region. Alternatively, if you're running Emacs under X Windows, you can use your mouse by pressing down on the left button at the start of the region and "drag" your mouse cursor to the end. To copy the region, use *[esc]-w*, while to cut the region, use *^w*. To paste, use *^y*. A variant of Emacs is Xemacs, which is essentially the same but with a more modern-looking user interface (buttons, menus, dialogs for opening files, etc). Nearly all keyboard commands that Emacs accepts are valid in Xemacs, too.

3 The Compilation Process

Before going into detail about the actual tools themselves, it is useful to review what happens during the construction of an executable program. There are actually two phases in the process, compilation and linking. The individual source files must first be compiled into object modules. These object modules contain a system dependent, relocatable representation of the program as described in the source file. The individual object modules are then linked together to produce a single executable file, which the system loader can use when the program is actually invoked. These phases are often combined with the *gcc* command, but it is quite useful to separate them when using make. For example, the command:

gcc -o prog main.c module1.c module2.c which produces an executable file *prog* can be broken down into the four steps of:

gcc -c main.c
gcc -c module1.c
gcc -c module2.c
gcc -o prog main.o module1.o module2.o



3.1 Compiler/Linker

Although there are a lot of different compilers out there, I "guarantee" that all of the problem sets can be solved using the GNU C Compiler, *gcc* to its friends. However, you can also use *g++*, which is just a wrap-around of *gcc*, to compile C++ programs. Using *g++* can have several advantages, it is pretty much ANSI compliant, available on a variety of different platforms and, more importantly, it works pretty reliably and directly compiles C, C++, and Objective-C.

3.2 Running *g++*

Even though it is called a compiler, *g++* is used as both a compiler and linker. The general form for invoking *g++* is : *g++ <option flags> <file list>*, where *<option flags>* is a list of command flags that control how the compiler works, and *<file list>* is a list of files, source or object, that *g++* is being directed to process. It is not, however, commonly invoked directly from the command line, that is what makefiles are for. If *g++* is unable to process the files correctly it will print error messages on standard error. Some of these error messages, however, will be caused by *g++* trying to recover from a previous error so it is best to try to tackle the errors in order.

3.3 Command-Line Options

Like almost all UNIX programs *gcc/g++* have myriad options that control almost every aspect of its actions. However, most of these options deal with system dependent features and do not concern us here. The most useful option flags for us are: *-c*, *-o*, *-g*, *-Wall*, *-I*, *-l*, and *-L*.

- *-c* Requests that *gcc* compile the specific source file directly into an object file without going through the linking stage. This is important for compiling only those files that have changed rather than the whole project.

  -o file Specifies that you want the compilers output to be named file. If this option is not specified, the default is to create a file 'a.out' if you are linking object files into an executable. If you are compiling a source file (with suffix .c for files written in C) into an object file, the default name for the object file is simply the original name with the '.c' replaced with '.o'. This option is generally only used for creating an application with a specific name (during linking), rather than for making the names of object files differ from the default source-filename .o.

  -g Directs the compiler to produce debugging information. I recommend that you always compile your source with this option set, since I encourage you to gain proficiency using the debugger (I recommend *gdb* or *ddd*). Note: The debugging information generated is for *gdb*, and could possibly cause problems with *dbx*. This is because there is typically more information stored for *gdb* that *dbx* will barf on. Additionally, on some systems, some MIPS based machines for example, this information cannot encode full symbol information and some debugger features may be unavailable.

  -Wall Give warnings about a lot of syntactically correct but dubious constructs. Think of this option as being a way to do a simple form of style checking. Again, I highly recommend that you compile your code with this option set. Most of the time the constructs that are flagged are actually incorrect usages, but there are occasionally instances where they are what you really want.

  Instead of simply ignoring these warnings there are simple workarounds for almost all of the warnings if you insist on doing things this way. This sort of contrived snippet is a commonly used construct in C to set and test a variable in as few lines as possible:

```
int flag;
if (flag = IsPrime(13)) {
...
}
```

  The compiler will give a warning about a possibly unintended assignment. This is because it is more common to have a boolean test in the if clause using the equality operator = rather than

to take advantage of the return value of the assignment operator. This snippet could better be written as:

```
int flag;
if ( (flag = IsPrime(13)) != 0) {
...
}
```

so that the test for the 0 value is made explicit. The code generated will be the same, and it will make us and the compiler happy at the same time. Alternately, you can enclose the entire test in another set of parentheses to indicate your intentions.

- *-Idir* Adds the directory *dir* to the list of directories searched for include files. This would be important for any additional files that I might give you on top of a given source code. There are a variety of standard directories that will be searched by the compiler by default, for standard library and system header files, but since we do not have root access we cannot just add our files to these locations. There is no space between the option flag and the directory name!

- *-llib* Search the library named *lib* for unresolved names when linking. The actual name of the file will be *liblib.a*, and must be found in either the default locations for libraries or in a directory added with the *-L* flag. The position of the *-l* flag in the option list is important because the linker will not go back to previously examined libraries to look for unresolved names. For example, if you are using a library that requires the math library it must appear before the math library on the command line otherwise a link error will be reported. Again, there is no space between the option flag and the library file name!

- *-Ldir* Adds the directory dir to the list of directories searched for library files specified by the *-l* flag. Here too, there is no space between the option flag and the library directory name.

3.4 Compiling in Emacs

The Emacs editor provides support for the compile process. To compile your code from Emacs, type *M-x compile*. You will be prompted for a compile command. The Emacs buffer will split at this point, and compile errors will be brought up in the newly created buffer. In order to go to the line where a compile error occurred, place the cursor on the line which contains the error message and hit *ˆc ˆc*. This will jump the cursor to the line in your code where the error occurred.

A Appendix: UNIX/Emacs Survival Guide

This appendix summarizes many of the commands helpful for getting around on the Unix operating system and the Emacs editor. All Emacs commands can usually be also for Xemacs which endows Emacs with a more modern user interface that allows for doing most actions with the mouse instead of the keyboard.

A.1 Directory Commands

cd <directory>	Change directory. If *directory* is not specified, go to home directory.
pwd	Show current directory (*print working directory*).
ls	Show (list) the contents of a directory. *ls -a* will also show files whose name begins with a dot. *ls -l* shows lots of miscellaneous info about each file.
rm <file>	Delete the file *file*.
mv <old> <new>	Rename a file from *old* to *new* (also works for moving things between directories). If there was already a file named *new*, its previous contents are lost.
cp <old> <new>	Creates a file named *new* containing the same thing as *old*. If there was already a file named *new*, its previous contents are lost.
mkdir <name>	Create a directory *name*.
rmdir <name>	Delete a directory *name*. The directory must be empty.

A.2 Shorthand Notations & Wildcards

.	Current directory.
..	Parent directory.
~	Your home directory.
~<user>	Home directory of *user*.
*	Any number of characters (not '.') Example: *.c is all files ending in '.c'.
?	Any single character (not '.').

A.3 Getting Help

man <subject>	Read the manual entry on a particular subject.
man -k <keyword>	Show all the manual listings for a particular keyword.

A.4 History

history	Show the most recent commands executed.
!!	Re-execute the last command.
!number	Re-execute a particular command by number.
!string	Re-execute the last command beginning with string.
Ctrl-P	Scroll backwards through previous commands.

A.5 Pipes

a > b	Redirect a's standard output to overwrite file b.
a >> b	Redirect as standard output to append to the file b.
a>& b	Redirect a's error output to overwrite file b.
a < b	Redirect as standard input to read from the file b.
a | b	Redirect as standard output to b's standard input.

A.6 Miscellaneous Commands

cat <file>	Print the contents of *file* to standard output.
more <file>	Same as *cat*, but only a page at a time (useful for displaying).
less <file>	Same as *more*, but with navigability (less is more).
w	Find out who is on the system and what they are doing.
ps	List all your currently active processes.
jobs	Show jobs that have been suspended.
process &	Runs a process in the background.
%	Continue last job suspended (suspend a process with ^Z).
% number	Continue a particular job.
kill <process-id>	Kill a process.
kill -9 <process-id>	Kill a process with extreme prejudice.
grep <expr> <files>	Search for an expression in a set of files.
wc <file>	Count words, lines, and characters in a *file*.
script	Start saving everything that happens in a file. Type *exit* when done.
lpr <file>	Print *file* to the default printer.
lpr -Pinky <file>	Print file to the printer named *inky*.
diff <file1> <file2>	Show the differences between two files.
telnet <hostname>	Log on to another machine.

B Appendix: GNU Emacs

For the following "^z" means hit the "z" key while holding down the "ctrl" key. "M-z" means hit the "z" key while hitting the "META" or after hitting the "ESC" key.

B.1 Running Emacs

emacs <filename>	Run Emacs (on a particular file). Make sure you don't already have an Emacs job running which you can just revive. Adding a "&" after the above command will run EMACS in the background, freeing up your shell.
^z	Suspend Emacs – revive with % command above.
^x^c	Quit Emacs.
^x^f	Load a new file into Emacs.
^x^v	Load a new file into Emacs and unload previous file.
^x^s	Save the file.
^x^k	Kill a buffer.

B.2 Searching

^s	Search for a string.
^r	Search for a string backwards from the cursor (quit both of these with ^f).
M-%	Search-and-replace.

B.3 Deletions

^d	Delete letter under the cursor.
^k	Kill from the cursor all the way to the right.
^y	Yanks back all the last kills. Using the ^x^c combination you can get a cut-paste effect to move text around.

B.4 Moving About

^f	Move forward one character.
^b	Move backward one character.
^n	Move to next line.
^p	Move to previous line (Note: the standard arrow keys also usually work).
^a	Move to beginning of line.
^e	Move to end of line.
^y	Scroll down a page.
M-y	Scroll up a page.
M-<	Move to beginning of document.
^x-[	Move to beginning of page.
M->	Move to end of document.
^x-]	Move to end of page.
^l	Redraw screen centered at line under the cursor.
^x-o	Move to other screen.
^x-b	Switch to another buffer.

B.5 Regions

Emacs defines a region as the space between the *mark* and the *point*. A mark is set with ^<spc> (control-spacebar). The point is at the cursor position.

M-w	Copy the region.
^w	Kill the region. Using ^y will also yank back the last region killed or copied. This is what we used for "paste" back in the bad old mainframe days before there was "paste".

B.6 Screen Splitting

^x-2	Split screen horizontally.
^x-3	Split screen vertically.
^x-1	Make active window the only screen.
^x-0	Make other window the only screen.

B.7 Miscellaneous

M-$	Check spelling of word at the cursor.
^g	In most contexts, cancel, stop, go back to normal command.
M-x goto-line <#>	Goes to the given line number.
^x-u	Undo.
M-x shell	Start a shell within Emacs.

B.8 Compiling

M-x compile	Compile code in active window. Easiest if you have a makefile set up.
^c^c	Do this with the cursor in the compile window, scrolls to the next compiler error. YEAH!

B.9 Getting Help

^h	Emacs help.
^h t	Run the Emacs tutorial.

Emacs does command completion for you. Typing M-x <spc> will give you a list of Emacs commands. There is also a man page on Emacs. Type *man emacs* in a shell.