As with Chapter 1, we are going to lay out the learning objectives for this chapter, and split them into what you should get on the first reading vs the second reading.

Note that this guide covers Appendecies A (quite short, and you probably need to read it only once), B.1–B.5, B.12, C.1–C.7, and D.4 together. This is the material that you need to get started on the Mastery Learning Platform.

The material in B.6 through B.11 is a bit more advanced use of the command shell and is not as critical right now. You will, however, want to master these skills eventually, as they will make a variety of tasks much simpler at the command line. Likewise, C.8 deals with programming commands, so we are going to wait until you are ready to write code before covering that. C.9 and C.10 cover more advanced Emacs features, which you will want to master later. (C.11 and C.12 are optional reading if you are interested at some point).

Appendix B also has a vareity of small activities (in green boxes) to try out at the command line. We recommend you do these on your second reading (not your first), and that in between your first and second reading (in addition to sleeping at least one night), you login to the Linux system for your class.

1 First Reading: Understand Key Ideas

On your first reading, the goal is to get the big ideas, and build the mental scaffolding in which to hold the details on your second reading—the learning objectives for your first reading are generally **Remember** and **Understand** objectives. If you are missing a few details (but not completely lost), it is ok to move ahead, and come back to them the next time through—you might ask questions on the course forums too! Note you should **sleep at least one night between the first and second reading** to give your brain time to process the information.

- **Expert Tools**
  - **Explain** the tradeoffs between *expert tools* and *novice tools*.
  - **Explain** how your tool selection can send signals about your expertise or lack thereof.
  - Note: you probably only need one reading of the Expert Tools Overview intro—it does not have much complex material.
- **In the Beginning Was the Command Line**
  - **Define** *command line*.
  - **Explain** how to get to a command prompt on the type of computer you use.
    - Note: since AoP was written, Windows has added "PowerShell" and "Windows Subsystem for Linux". These are both much better options than what previously existed. WSL will give you an Ubuntu environment on your Windows computer.
  - **Explain** why a commmand prompt might display the username and hostname.
- **Getting Help**
  - **Define** *command line arguments*.
  - **Explain** options.

- **State** what the `man` command does.
- **List** the basic commands to use to scroll and quit in `man`.
- **Explain** why there are sections for the man pages.
- **State** which section of the man pages is for Linux commands and which section is for C library functions.
- **Explain** how you would find information about the man pages, including other section numbers.
- **State** how to keyword search the man pages.
- Note: while AoP does not explicitly discuss this, one important thing to remember is that man pages are *authoritatative documentation*—they were written by experts, reviewed by experts, and the chances of incorrect information in them is quite low. By contrast, you must be incredibly careful of whatever you find on the Internet—anyone can post things on the Internet, and there are significant chances of them being wrong. In some cases, what you find might be wrong in subtle and hard to tell ways. This warning is especially applicable of StackOverflow, where it can be quite hard for a novice to distinguish correct from incorrect information.

- **Directories**
  - **Define** *filesystem*.
  - **Define** *directory*.
  - **Explain** when it is appropriate to use the term "folder" (and try to avoid it as much as possible, in favor of "directory").
  - **Define** *root directory*.
  - **Explain** the organization of a Unix filesystem.
  - **Define** *path*.
  - **Define** *absolute path*.
  - **Define** *current directory* (also known as "current working directory").
  - **Define** *relative path*.
  - **Identify** whether a path is absolute or relative.
  - **Define** *home directory*.
  - **State** what `~` is short for in Unix.
  - **Explain** how to change the current directory of your command shell.
  - **Explain** how to list the files in the current directory.
  - **Explain** how to list the files in directories other than the current directory.
  - **Explain** the significance of filenames starting with a dot (`.`) and how to make ls show them.
  - **State** what the special directory names `.` and `..` signifiy.
  - **Explain** how you would find more information about the ls command.
  - **State** what the `mkdir` command does.

  - **State** what the `rmdir` command does.

- **Displaying Files**
  - **Explain** what the `cat` command does and how to use it.
  - **Define** *standard input*.
  - **Explain** what the `more` command does and how to use it.
  - **Explain** what the `less` command does and how to use it.
  - **Explain** what the `head` command does and how to use it.
  - **Explain** what the `tail` command does and how to use it.

- **Moving, Copying, and Deleting**
  - **Explain** what the `mv` command does and how to use it.
  - **Explain** what the `cp` command does and how to use it.
  - **Explain** what the `rm` command does and how to use it.

- Note: remember we are coming back to B.6–B.11 later. It will be helpful for you to practice with the basics. For now, skip ahead to B.12.

- **Remote Login: ssh**
  - **Define** *ssh*.
  - **Explain** how to ssh into a server from your computer.
    - Note: Windows Powershell and WSL support ssh directly from their command lines like Mac and Linux. Also, since the writing of AoP, MobaXTerm has become a more popular Windows alternative if not using Powershell or WSL.
  - **Explain** why ssh is more secure than other login methods.
  - **Explain** what scp is for and how to use it.

- Note: we are going to continue directly to Emacs (Appendix C) here. However, if you want to end your first reading of Appendix B, come back to it, and practice before moving on, you are more than welcome to. You may also continue on through Appendix C—it is a matter of personal choice, and how you are feeling about the cognitive load of what you have read so far.

- **Emacs**
  - **Define** *editor*.
  - **Define** *In The Zone*.
  - **Explain** why/how use of the mouse inhibits effective programming.
  - **Define** *muscle memory*.
  - **Define** *syntax highlighting*.
  - **Explain** why automatic indentation is useful.

- **Explain** why it is important for your editor to integrate with other programming tools.
- **Explain** why it is important to use only one editor for all programming languages you work in and most other tasks you do.

- **Emacs Vocabulary**
  - **Define** *buffer*.
  - **Define** *frame*.
  - **Define** *window*.
  - **Define** *point*.
  - **Define** *mark*.
  - **Define** *C-(key)*.
  - **Define** *M-(key)*.
  - **Define** *kill*.
  - **Define** *yank*.
  - **Define** *extended command*.
  - **Define** *RET*.
  - **Define** *Minibuffer*.
  - **Define** *Major mode*.
  - **Define** *Minor mode*.
  - **Define** *Balanced Expression*.
  - **Define** *Modeline*.

- **Running Emacs**
  - **Explain** how to run emacs

- **Files and Buffers**
  - **Explain** how to perform basic editing in Emacs.
  - **Learn** the keystrokes for the basic commands in Table C.1
    - Note: we do not expect you to memorize them (memorization is bad). However, you should become acquainted with them fairly quickly as you use them often. You want your "fingers to learn" (muscle memory) these commands. We note this learning objective here because we recommend you start making a quick reference sheet with these (and other commands) to work from as you learn them. Try to learn a couple Emacs commands per day.
  - **Interpret** Emacs keystrokes, recognizing the difference between *e.g.*, `C-a C-b` and `C-a b`.
  - **Recognize** that Emacs has TAB completion, and know that you should make use of it frequently.
  - **Explain** how to suspend Emacs, and restore it to the foreground.

  - **Learn** the commands for cancelling and undoing (Table C.2)
    * Again, don't memorize: put them on your quick reference sheet and practice them until they go into muscle memory.
  - **Explain** how to redo (un-undo) in Emacs.
  - **Explain** undo-in-region
    * Note: this feature is INCREDIBLY useful.

- **Cut, Copy, and Paste**
  - **Explain** how to copy/paste in Emacs.
  - **Learn** the commands for copy/paste in tables C.3 and C.4.
    * Again, don't memorize: put them on your quick reference sheet and practice them until they go into muscle memory.
  - **Explain** the idea of the kill ring, and how to paste something other than the most recent copy/cut
    * Note: pasting from further back in the kill ring is another INCREDIBLY useful feature.

- **Multiple Buffers**
  - **Explain** why you might want to see multiple buffers at once.
  - **Learn** the commands from C.5
    * Again, don't memorize: put them on your quick reference sheet and practice them until they go into muscle memory.

- **Search and Replace**
  - **Define** *incremental search*.
    * Note: incremental search is also quite useful for lightweight fast cursor movement!
  - **Learn** the commands from C.6
    * Again, don't memorize: put them on your quick reference sheet and practice them until they go into muscle memory.

- Note: we will wait for C.8–C.10 until you are ready to start writing code. C.11 and C.12 are optional if you are interested. Please go to D.4 (git) next.

- **Revision Control: Git**
  - **Define** *revision control*.
  - **Explain** why you want revision control in software projects (and in so many other things!).
  - **Recognize** that Git is a revision control system.
  - **State** where you can go for a lot more in-depth information on Git.
  - **Explain** what happens when you *commit*, and what benefits a commit provides.

- **Explain** why you do not want to just manually keep copies of your code.
- **Define** *pull*.
- **Define** *push*.
- **Explain** how revision control supports collaboration.
- **Explain** how/why there might be many "current" versions of a piece of code.

2 Second Reading: Work Through Examples

These Appendecies are a bit different from the main chapters, in that they provide a bunch of skills to practice with. The higher level learning objectives are basically all **perform** (the right Linux, Emacs, or Git command. Instead of listing these all out, we are going to give you an outline for how to go back through and practice:

1. Make sure you have access to a UNIX command line (either by using a Mac or Linux computer, by using WSL on Windows, or by SSHing to a Linux server). For anyone reading this for ECE 551, MEMS 555, or GameDev 510, you will have access to a Linux server for class, so we recommend working there. See B.12 as well as information from your specific course for how to login (and ask for help if you need it!)

2. Re-read Appendix B.1–B.5 and try the activities in the green boxes.

3. Do MLP assignment `000_submit` which will help you practice some basic Emacs and Git use.

4. Open a file in Emacs, and re-read Appendix C.1–C.7. Try out the various commands as you read about them. Ask questions about any commands you don't understand. Make notes of which commands you think you need to learn most quickly (will use most often). Remember you will be using Emacs a lot, and have a lot of time to practice with commands. The sooner you become fluent in the commands you need frequently, the easier your editing will be.

    - Note: if there is something you want to do in Emacs, or some way you want it configured, please ask. If your instructor doesn't know, they can ask Drew, who is more than happy to tell you (or find out if he doesn't know).

5. Re-read D.4 now that you have practiced the basics of git. Note that we will only be using the basics of git in this course. You might use more advanced git in later courses. Also, please note that you do not need to use more advanced git on the MLP. Many students have made messes of their git repositories by using git rebase, init, or other advanced commands "because someone said to on StackOverflow" without understanding what those commands did.