Slide 1

Administrivia

- Peer tutoring available for this class, organized by student ACM chapter (ACM is the major professional organization for CS). Hours this semester are M/T/W/R 5pm to 9pm in 270A and 270L, F and weekend by appointment, starting next week. Great resource for getting help with homeworks!
- (Status check — does the bookstore have copies of the text yet?)

Slide 2

Minute Essay From Last Lecture

- Several people mentioned being surprised by how much you can apparently do from the command line. I found this surprise surprising! but then I started out with command lines.

  It should make perfect sense, though, if you keep in mind that UNIX dates back to an era in which command-line interfaces were the norm, and it still tends to attract people who like this interface.

Slide 3

Review — Commands For Working With Files and Directories

- `cat`, `less` to display files.
- `cp`, `mv`, `rm` to copy, move/rename, remove files. `-i` to prompt (`rm`) or warn about overwrites (others). (Why isn't this the default behavior? System was designed to be expert-friendly and so assumes you meant what you said, maybe.)
- `mkdir`, `rmdir` to create, remove directories.
- `cd` to move between directories. `ls` to display files in directory (`-l` for long format, `-A` to also show hidden files, `-d` to show information about directory itself.)

Slide 4

Other Useful Commands

- `man` *command* to get information ("man page") about *command* Also displays information about C-library functions.

  Reference information rather than a tutorial, but usually very complete.

  Sometimes there are multiple man pages with the same name (e.g., a command and a function); `man -a` to get all of them (`q` to move from one to the next).

  `man -k` *keyword* to look for commands that might have something to do with *keyword*.
- `man` uses `less` to page through documentation. Up and down arrows work to move through file. `/` searches for text in file. `q` exits. `h` shows list of other options.

Slide 5

Text Editors — Review

- "Text editor" is a program for creating and editing plain text (as opposed to, e.g., a word processor).
- I use and will show in this class `vim`. Not especially beginner-friendly but (IMO!) "expert"-friendly, and good for working with program source code.
- I showed a few basics last time. More in Dr. Lewis's video lectures and my "notes" listed under readings. Consider making yourself a "cheat sheet" of tips that sound useful to you, so you'll remember them when using this editor. Using this editor is apt to be tedious if all you remember is the bare minimum.

Slide 6

A Little About Shell Customization

- Can be very useful to customize your shell a bit — e.g., to always use those `-i` flags.
- To do this, edit file `.bashrc` ...

  No. First save old file (`cp .bashrc save.bashrc`), so if you really mess up you can get the old one back.

  Now edit `.bashrc` (with `vim` — "of course"?) and add lines such as

  ```
  alias cp='cp -i'
  alias mv='mv -i'
  ```

- Save, quit, *open new terminal window* (leave the old one open in case you messed up), and if you type `which cp` you should see your alias. (If something goes wrong, in old terminal window type `cp save.bashrc .bashrc` to restore.)

Slide 7

Input/Output Redirection in UNIX/Linux

- A key feature of command-line environments, one that provides a lot of power, is "I/O redirection". Idea is that programs can get input from different sources (keyboard, file, "pipe") and write output to different destinations (terminal, file, "pipe"), all without changing the program. Example:

  ```
  myprogram < test1-in > test1-out
  ```

  to have `myprogram` get its input from `test1-in` rather than the keyboard, and put its output in `test1-out` rather than showing it on the screen. (Overwrites `test1-out`. To append instead, use `>> test1-out`.)

  This is (part of) how I grade programming homework!

- "Pipes" connect output of one program with input of another. A common "use case" is to page through long output by piping it into `less` — e.g.

  ```
  ps aux | less
  ```

Slide 8

"Why C?" Revisited

- Recently we asked ENGR again about what language to use in this course. Their answer indicated that at least some students will use C in a project at some point. Good to know?

Slide 9

A First Program in C

- As you read sections of the textbook you may want to try running the programs yourself. More about all of this soon, but today let's do a "hello world" program . . .
- ("Hello world" program? Yes. Traditional in some circles to have one's first program in a language print "hello, world" to "the screen". Origins of this tradition — inventors of C!)

Slide 10

A First Program in C, Continued

- First write the program using a text editor (e.g., `vim`) and save it with a name ending in `.c` (say `hello.c`). (See the "sample programs" Web page for what it looks like.)
- Next, compile the program (turn it into something the computer can execute). Simplest command for that:

  ```
  gcc hello.c
  ```

  If no syntax or other errors, produces an "executable" file `a.out`.
- Run the program by typing `./a.out` at the command prompt.

Slide 11

A Little More About the Program

- Almost everything in the simple program is standard boilerplate that all your programs will have. We'll talk more later about what it all means. For now focus on the single line with `printf`; this is the one that accomplishes the program's purpose.
- With what we know now, we can't write programs that are much more interesting, though we could put in some more lines using `printf`. Try that? Note that even this small addition illustrates something important: Unless otherwise indicated, the computer executes code "sequentially" (in the order in which it appears in the source).

Slide 12

Programming Basics and C

- Previous lecture described relationship between what humans write ("source code") and what computers execute ("machine language").
- For traditional "compiled languages" such as C, source code must be transformed not just into machine language, but into a complete "executable file" (machine language for your code, plus machine language for any library functions, plus information so operating system can load it into RAM and start it up. (Detail: This is for "hosted environment"; in some environments in which C is used, there may be no O/S.)
- So, what happens to your code . . .

Slide 13

Programming Basics and C, Continued

- Your code is first "compiled" into "object code" (machine language).

  Then it's "linked" with any library object code to form "executable file".

  Sometimes (as for examples we're doing now) both steps happen as a result of a single command.
- To recap, we're basically using two tools, `vim` (to write/edit programs) and `gcc` to compile them, and a command-line environment (terminal window) to run both tools and also programs we write.

Slide 14

Minute Essay

- Students sometimes mention that what I do in class moves fast. Too fast? I feel like you learn this stuff better by practicing/exploring at your own pace outside class, but . . . ?
- Questions?