Slide 1

Administrivia

- (None?)

Slide 2

Minute Essay From Last Lecture

- Most people said the Linux command line seems to be at least a little similar to Matlab. Seems plausible — I think all command-line interfaces (CLIs for short) are similar in some ways.
- Is this CLI "user friendly" . . . Eh. "Choosy about its friends"? Or "expert friendly" as (sometimes) opposed to "novice friendly". (A tool could be both, but that seems rare, alas.)
- Names of commands are cryptic, yes. Invented in a time when short names were very desirable, and then people got used to them.
- Commands aren't control-something . . . Right. This makes sense if you know that they're names of executable files, usually in some system directory somewhere.

Slide 3

Useful Command-Line Tips

- Pedantic aside: "The shell" here means the one you're most likely to be using. There are other programs with similar functionality you could use instead.
- The shell (the application that's processing what you type) keeps a history of commands you've recently typed. Up and down arrows let you cycle through this history and reuse commands. You can even search the history (by pressing ctrl-r and then typing the text to search for).
- The shell offers "tab completion" for filenames — if you type part of a filename and press the tab key, it will try to complete it.
- Together these can save you a lot of typing!
- If you have trouble remembering the commands (which you likely will at first!): In times past beginners got paper "cheat sheets" of commonly-used commands. Maybe make yourself an electronic equivalent?

Slide 4

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.)

Slide 5

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 open `.bashrc` 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 say `cp save.bashrc .bashrc` to restore.)

Slide 6

Other Useful Commands

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

  This is 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 7

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.
- Start `vim` with `vim` *filename*. Can only enter text in "insert mode". Start with `i` or `a`. Exit with ESC.

Slide 8

`vim` Tips

- Biggest hurdle may be the notion of modes. (But you already know about this, sort of? Word processors have insert/overwrite modes.)
- Cut/copy/paste basics:

  `dd` cuts a whole line. `yy` copies a whole line.

  `p` pastes after the current line. `P` pastes before the current line.
- Search by typing `/`, text to search for, Enter. Repeat search with `n`. Search-and-replace using this, `cw`, and `.`

Slide 9

More `vim` Tips

- `:help` brings up online help. `:help visual-mode` describes one feature you may like.
- `u` to undo. `:w` ("write") to save. `:q` to exit. `:q!` to exit without saving.

Slide 10

`vim` Tips — Errors/Mistakes

- If you type just `q` rather than `:q`, `vim` thinks you want to record a macro. Screen will show "recording". Press `q` to make it stop.
- If you type `q:` rather than `:q`, `vim` thinks you want it to display a history of commands and shows them to you in a subwindow. Type `:q` to make that go away.
- If you want to copy-and-paste text using window manager, `:set paste` first to avoid annoying indentation behavior. `:set nopaste` after.

Slide 11

vim Tips — Errors/Mistakes, Continued

- If you just close the terminal window when running vim, that "crashes" vim. So what? Well . . .
- vim creates a hidden file that saves information that can help with recovery if it crashes. Deleted on normal exit, otherwise not. And then next time you start vim on that file — screenful of messages starting "ATTENTION" and "Found a swap file" and finally asking you whether you want to open it anyway or what. If you respond R vim will try to recover unsaved changes; otherwise not. To actually delete this hidden file, so you don't get that same screenful of messages next time, respond D.

Slide 12

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 ("screen", 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 13

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 14

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 15

Minute Essay

- One person said Monday's class moved kind of fast. Too fast? I feel like you learn this stuff better by practicing/exploring at your own pace outside class, but . . . ?
- Questions? (We'll start talking next time about what those lines in the program mean.)
- (Thanks, by the way, for helping me by using subject lines as asked for — "minute essay" and "1312" or "cs1".)