Slide 1

Administrivia

- Update on book: Publisher will allow us to give you online access to early chapters through Google Docs. You should have an e-mail about this from me, with a link you click to get access.
- I will also post links to the videos Dr. Lewis is making for the course.
- If you're a prospective major/minor, or if you just want to know about departmental events, consider signing up for our CSMajors mailing list: Go to the department's (old!) Web page at `http://www.cs.trinity.edu` and follow the link that says "Subscribe . . . ".)

Slide 2

More Administrivia

- Homework 1 on Web; due next Wednesday. (Should be fairly easy — practice with tool(s).)
- A request: You will turn in most if not all work for this course by e-mail. *Please* include the name or number of the course in the subject line of your message, plus something about which assignment it is, to help me get it into the correct folder for grading.

Slide 3

Steps in Solving Problems on a Computer

- Understand the problem — what do you want the computer to do, exactly?
- Design a solution suitable for a computer ("develop an algorithm").
- Implement the solution ("write the program"). This requires expressing your ideas in "a programming language" — of which there are many! Programming languages are similar to human languages in some ways, different in others; they are meant to be (somewhat!) human-readable while still being precise enough for a computer to understand.
- Test your solution. This will involve the use of some tool that translates what you write ("source code") into something the computer hardware can work with.

Slide 4

Solving Problems on a Computer, Continued

- The overall process — understand the problem, develop and test a solution — is mostly independent of the choice of programming language and platform (combination of hardware and operating system, roughly). So once you understand the principles it is relatively easy to learn new languages.
- Opinions about which language to learn first, and on what platform, vary. Right now different sections of this course use different languages but the same platform (Linux). For this course we will use Scala; it is somewhat easier for beginners than some of the other choices but also powerful enough to write interesting programs.

Slide 5

Programming Basics

- What computers actually execute is *machine language* — binary numbers each representing one primitive operation. Once upon a time, people programmed by writing machine language (!).
- Obviously that was tedious and error-prone. A very early bright idea — write something more human-readable (*source code*) and *have the computer translate it*. Useful even if the source code is just a human-readable version of the primitive operations (*assembler language*). Even better if the source code is less primitive (*high-level language*).
- Source code is simply plain text (as opposed to text plus formatting, as in a word-processor document). Since the hardware doesn't understand it, however, . . .

Slide 6

Programming Basics, Continued

- Source code can be *interpreted* — translated line by line into something the hardware can understand, by another program called an *interpreter*.

  (This is how "scripting languages" work. An example is the command shell's language. !)
- Or it can be *compiled* — translated by a program called a *compiler* into something the hardware can execute directly.

  (This is how traditional "high-level" languages such as C and Fortran work.)
- Or it can be compiled into some intermediate form that can be executed by another program.

  (This is how some recent languages such as Java work.)

Slide 7

Writing Source Code

- How do you get source code? If using an interpreter, you can just type it in. If you want something you can keep and reuse, however, you need a tool that will do that.
- Simplest way to create source code is with a *text editor* — a program for writing and editing plain text. This is what we will do for now.
- (Another way is to use an *IDE* (Interactive Development Environment). We will try one of these later in the semester.)

Slide 8

A Word About Tools

- In this class we use Linux and command-line tools because we believe it is important for budding computer scientists to know how to work with these tools.

  For others — exposure to something new and different?
- (What is Linux? it's an operating system, as Windows and Mac OS X are operating systems. It's one of a family of operating systems descended from UNIX, developed at Bell Labs in the early 1970s. A lot of servers run Linux or some other UNIX-like system. There are also ongoing efforts to develop mainstream desktop systems.)
- A UNIX person's response to claims that UNIX isn't user-friendly: "Sure it is. It's just choosy about its friends."

Slide 9

Getting Started with Linux

- When you log in, you should get a graphical desktop, which should be navigable with what you know from using other graphical environments (though some details are different).
- In Linux, we talk about files and directories; the idea is the same as Windows' files and folders, though again some details are different.
- The graphical system should give you a way to get a terminal window. Once you have that . . .

Slide 10

Getting Started With the Command Line

- What you get when you start a terminal window is a "command shell", similar to Windows' "MS-DOS prompt".

  Rather than pointing and clicking, you type the name of the program you want to run, plus whatever arguments (parameters) it needs.
- (Why would you want to use a command line? because for some things it's arguably more efficient, and it's "scriptable" in ways that GUIs typically aren't.)
- Let's try some commands . . . (Don't worry if this goes by quickly — you should plan anyway to spend some time outside class trying out what we do in class and what's in the reading.)

Slide 11

Some Commands

- `pwd` shows the current directory. (When you give a filename, it's relative to this directory unless you give a full pathname.)
- `ls` lists the current directory. Add `-l` to get more information.
- `cd foo` changes to directory `foo`. Just `cd` goes back to your home directory. Try `cd Local` and then `ls`.
- `mkdir foo` creates a director `foo`. Might be useful to create one for your files for this class.
- `passwd` changes your password. (Not a command you'll want often, but probably now!)

Slide 12

Useful Command-Line Tips

- 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.

  (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 offers "tab completion" for filenames — if you type part of a filename and press the tab key, it will try to complete it.
- To learn more about command `foo`, type `man foo`. This is reference information rather than a tutorial, but usually very complete. `man -k foo` will give you a list of commands having something to do with `foo`.

Slide 13

Remote Access

- One of the strengths of a command-line enviroment is that it works well for "remote access" (using the computer when you aren't sitting in front of it).
- To do this from another UNIX-like computer, use `ssh`. `scp` and `sftp` can be used to copy files.
- From a Windows computer, install either Cygwin (UNIX-like toolkit) or PuTTY (terminal emulator).
- More details in chapter 2 of book and on the course Web site ("Useful links" to "More useful links for 1320/1321" to "More about remote access").

Slide 14

Text Editors

- Many, many text editors, and people have favorites. Notepad is an example from the Windows world.
- I use and will teach in this class `vi`: It's found on every UNIX/Linux system I know of, and is very powerful, though it takes some getting used to. (`vi` on our Linux machines is actually `vim`, a more capable "clone" of the original `vi`.)
- Other popular Linux text editors include `emacs`, `pico`, and `gedit`. Advice: Give `vi` a real try first, but if using it is just too painful, use something else!

Slide 15

`vi` Basics

- `vi` has two *modes* — insert mode (where what you type goes into the file) and command mode (where you can type commands to copy, move, delete, save, etc.).
- You start an editing session by typing, e.g., `vi hello.txt`. It starts in command mode. Enter insert mode by typing `i`. Exit by pressing ESC. Move around with the arrow keys. (Try entering some text.) Delete a single character with `x` (in command mode).
- Save and exit by typing `:wq`.
- *Highly recommended:* `vimtutor` brings up an interactive tutorial. (Homework 1 asks you to try it.)

Slide 16

`vi` 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 `.` (See book.)

Slide 17

`vi` Tips — Errors/Mistakes

- `u` means "undo" the previous action (insertion, deletion, paste). Repeat to undo multiple actions.
- `:q!` exits without saving. Useful if you make a complete mess of things.

Slide 18

More Commands

- Now that we have a way of creating files, we can try out some other basic commands.
- `cat` to show contents of a file. `more` or `less` to show it a screenful at a time.
- `cp` to copy one file to another. `-i` to warn about overwrites.
- `mv` to move or rename a file. `-i` to warn about overwrites.
- `rm` to delete a file. (Note — no recycle bin, so use with caution! or `-i` to prompt.)
- Other useful/interesting commands in chapter 2. Good to go through the list and try them out for yourself.

Slide 19

UNIX Filesystem Basics

- Unlike in Windows (and Mac?), UNIX filesystems are case-sensitive (so hello and Hello are different files).
- Files have two levels of ownership — "owner" (user) and "group". Groups allow sharing files with some but not all users.
- File access is controlled by "permissions". Three levels (owner, group, and everyone else), three types of access (read, write, execute).
- `ls -l` shows permissions. `chmod` changes them.

Slide 20

Minute Essay

- Anything today that was particularly unclear?