Slide 1

Administrivia

- Accounts and passwords should be set up. (We will try to straighten out lost/missing passwords in class.) To change your password, use the command `yppasswd`. *This didn't work in class. Stay tuned.*
- Information about Unix/Linux commands, text editors, etc., is available via the "Useful links" page (here).

Slide 2

A Word About Tools

- In this class we use the C programming language, Linux, and command-line tools, because we believe it is important for budding computer scientists to know something about "low-level computing".

  For others — exposure to something new and different, and a good foundation for learning whatever other programming language you might later want to use.
- A Unix person's response to claims that Unix isn't user friendly: "Sure it is. It's just choosy about its friends."

Slide 3

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 4

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.
- Let's try some:
  - `pwd` shows the current directory.
  - `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`.

Slide 5

Useful Command-Line Tips

- The shell keeps a history of commands you've recently typed. Up and down arrows let you cycle through this history and reuse commands.
- 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 also works with C library routines — more about them later.) This is reference information rather than a tutorial, but usually very complete.

Slide 6

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 (!).
- Now, "programming" as we will use it means writing *source code* in a *high-level language*.
- Source code is simply plain text, which is *compiled* into *object code* (machine language) and then *linked* with other object code (including system libraries) to form an *executable* (something the operating system can execute).

Slide 7

Source Code and Text Editors

- How do you get source code?
- The simplest way is to create it with a *text editor* — a program for writing and editing plain text.
- 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 a little 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`.

Slide 8

`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.c`. It starts in command mode. Enter insert mode by typing `i`. Exit by pressing `ESC`. Move around with the arrow keys. Delete a single character with `x`. (Try entering some text.)
- Save and exit by typing `:wq`.
- *Highly recommended:* `vimtutor` brings up an interactive tutorial.

Slide 9

A First C Program

- Let's write the traditional "hello world" program in C, using `vi`.
- Once it's written, compile-and-link by typing `gcc hello.c`. (There are other options you should use, but for now this is okay.) Result is `a.out`.
- Execute by typing `a.out`.

Slide 10

More Commands

- Now that we have a couple of files, we can try out some other basic commands.
- `cp` to copy one file to another.
- `mv` to move or rename a file.
- `rm` to delete a file. (Note — no recycle bin, so use with caution.)

Slide 11

Structure of a C Program

- Pre-processor directives: These begin with # and are used to (among other things) include in the compilation process information about libraries.
- Global identifiers (functions and variables). Function declarations here are often useful; variables are usually bad practice.
- Function(s), possibly containing variables, returning values, etc. More about all of this later.

Slide 12

Comments

- Anywhere in the program you can include *comments*, meant for human readers and ignored by the compiler. The old C style is to start with /* and end with */. The book shows a newer style.
- I will nag you a lot about putting in good comments: They can be very useful for human readers (me now, you if you look at your programs again in a year, someone else if you program as part of a team, etc.).

  Also can be a good way to "think out loud" about a program before starting to code.

Minute Essay

- What today seemed most unclear?
- (Informal assignment for next time: Try to do what we did in class today on your own.)

Slide 13