Slide 1

Administrivia

- Reminder: Homework 1 due Tuesday (at 11:59pm).

Slide 2

Minute Essay From Last Lecture

- A lot to learn — yes, it is rather. Not unusual when tackling a new subject/activity?
- No good way to remember all the commands — yeah, it's one of the drawbacks of a CLI, but you're apt to remember the ones you use most. In times past beginners got paper "cheat sheets" of commonly-used commands. Maybe make yourself an electronic equivalent?
- Surprising that text editor opens in the terminal window rather than making a new window — yes, maybe! but `vi` is one of a group of "text-mode" programs originally designed to run on plain-text terminals.

Slide 3

Command Line Review

- Last time we looked at commands for navigating the file system and working with files (moving, copying, etc.). Other useful/interesting commands in chapter 2. Good to go through the list and try them out for yourself.

  (Yes, if you're sitting in front of the machine you can use the GUI. If you're logged in from somewhere else, the command line may work better.)
- Remember/note that `man` shows you information about a command, and `man -k` shows you a list of commands related to a keyword.

Slide 4

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 5

Input/Output Redirection

- Normally programs run from the command line write output to the terminal window. Can instead "redirect" output to a file:

  `> outfile` (overwrite)

  `>> outfile` (append)

- Normally programs get input from the keyboard, but can also make them get input from a file with `<`. More later.

- Finally, can use "pipes" (vertical-bar `|`) to have output from one program become input to another. Example:

  `ruptime | less` (show status of lab machines)

  Very powerful idea! this and some other ways of connecting simple programs makes for a very powerful and flexible environment.

Slide 6

`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 7

`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 8

Scala

- Scala is short for "scalable programming language". (We may talk more later about what that means.) Relatively new language, but we think good for a first course.
- Various options for running Scala source code. Today we will look at two of them — typing it in interactively, and executing "scripts".

  `scala` starts an interactive environment ("REPL" – "read, evaluate, print" loop), good for trying things out.

  `scala program.scala` runs the program in file `program.scala`.
- By tradition (established by the inventors of the C language, in 1970-something), our first program will just write to the screen "hello, world").

Slide 9

A First Scala Program

- Type `scala` at the command line to start the interpreter.

  Now type

  ```
  println("hello, world")
  ```

  (and press return).

  Try misspellings and other variations. Type `:quit` to end.

- Or we could put that single line in a file `hello.scala` and run it with the command

  ```
  scala hello.scala
  ```

Slide 10

Comparison — Python

- An equivalent program in Python (the language being used in some other sections of POP I):

  ```
  print "hello, world"
  ```

- Run interactively or as script using command `python`.

Slide 11

Comparison — C

- An equivalent program in C (the language previously used in POP I, when it was PAD I):

```
#include <stdio.h>

int main(void) {
    printf("hello, world\n");
    return 0;
}
```

- No option for running interactively; first compile (command `gcc`) to create executable file, then "run" the file.

Slide 12

Comparison — Java

- An equivalent program in Java (a language often used as a first language in high-school courses):

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

- No option for running interactively; first compile (command `javac`) to create Java bytecode, then run bytecode using command `java`.

Slide 13

Programming in Scala

- In Scala (as in many, maybe most, programming languages) two of the basic building blocks are expressions (similar to expressions in math) and statements (roughly speaking, complete instructions).
- We will talk more about this next time, but for now we can try a few things . . .
- Start up the Scala interpreter and try typing in a few arithmetic expressions.
- Try making the “hello world” program print a second line.

Slide 14

Minute Essay

- Anything today that was particularly unclear?
- Have you tried accessing the lab machines remotely? If so, how / from what kind of computer, and did it work?