Slide 1

Administrivia

- Homeworks 6 and 7 on Web. Nominally due next Monday, but will be accepted up to a week late without penalty. I'm planning one more homework (a not-too-tough "sum up what you learned" assignment).
- Info on projects on Web (linked from due date, May 10). Notice that proposals must be submitted by April 29.

Slide 2

Minute Essay From Last Lecture

- Question: What do you currently use to produce formatted documents? What do you like/dislike about it?
- Most frequently mentioned was MS Word. A few mentions of OpenOffice Writer, one of LaTeX.

  Noteworthy "don't like": Autoformatting.

  Noteworthy "like": "Universal format." (Only, of course, for platforms for which MS Word is available.)

Slide 3

The `make` Utility

- Motivation: Most programming languages allow you to compile programs in pieces ("separate compilation"). This makes sense when working on a large program — when you change something, just recompile the parts that are affected.
- Idea behind `make` — have the computer figure out what needs to be recompiled and issue the right commands to recompile it.

Slide 4

Makefiles

- First step in using `make` is to set up a "makefile" describing how the files that make up your program (source, object, executable, etc.) depend on each other and how to update the ones that are generated from others. Normally call this file `Makefile` or `makefile`.

  Very simple example:

```
main:   main.o foo.o
        gcc -o main main.o foo.o
main.o: main.c defs.h foo.h
        gcc -c main.c
foo.o:  foo.c
        gcc -c foo.c
```

- When you type `make`, `make` figures out (based on files' timestamps) which files need to be recreated and how to recreate them.

Slide 5

Useful Command-Line Options

- `make` without parameters makes the first "target" in the makefile. `make foo` makes `foo`.
- `make -n` just tells you what commands would be executed — a "dry run".
- `make -f otherfile` uses `otherfile` as the makefile.

Slide 6

Defining Rules

- You define dependencies for a rule by giving, for each "target", a list of files it depends on.
- You also give the list of commands to be used to recreate the target.

  NOTE!: Lines containing commands must start with a tab character. Alleged paraphrase from an article by Brian Kernighan on the origins of Unix:

  > The tab in makefile was one of my worst decisions, but I just wanted to do something quickly. By the time I wanted to change it, twelve (12) people were already using it, and I didn't want to disrupt so many people.

Slide 7

Phony Targets

- Normally targets are files to create (e.g., executables), but they don't have to be. So you can package up other things to do ...
- Example — many makefiles contain code to clean up, e.g.:

```
clean:
        -rm *.o main
```

To use — `make clean`.

Slide 8

Variables in Makefiles

- You can also define variables, e.g.:
  - List of object files needed to create an executable. Then use this list to specify dependencies, command.
  - Pathname for a command, options to be used for all compiles, etc.
- Example:

```
objs = main.o foo.o
CFLAGS = -Wall -pedantic
main:   $(objs)
        gcc $(CFLAGS) -o main $(objs)
```

Slide 9

Predefined Implicit Rules

- `make` already knows how to "make" some things — e.g., `foo` or `foo.o` from `foo.c`.
- In applying these rules, it makes use of some variables, which you can override.
- A simple but useful makefile might just contain:

```
CFLAGS = -Wall -pedantic -O
```

- Or you could use

```
CFLAGS = -Wall -pedantic $(OPT)
OPT = -O
```

and then optionally override the `-O` by saying, e.g., `make OPT=-g foo`.

Slide 10

Implicit Rules (Pattern Rules)

- You can define similar rules — e.g., a makefile to compile `.c` files using the MPI C compiler:

```
MPICC = /usr/bin/mpicc
CCFLAGS = -O -Wall -pedantic

%: %.c
        $(MPICC) -o $@ $(CCFLAGS) $<
```

`$<` is the `.c` file here (first prerequisite), and `$@` is the target.

(Note that this is for GNU `make`. Non-GNU `make` has a similar idea — "suffix rules" — with slightly different syntax.)

Slide 11

Other Uses For `make`

- `make` can be used to automate things other than compiling programs. It's particularly useful for defining implicit rules.

  Example: Makefiles to run `latex` and associated programs.

Slide 12

Minute Essay

- One more real lecture, plus some time on the last class day. We've covered most of the material I wanted to be sure to include. What to do in the remaining time?

  Possible topics:

  - Useful/standard Unix utilities (e.g., `tar`).
  - Text-based mail programs and/or mail filtering with `procmail`.
  - CGI scripts.
  - A little about X, desktop environments, window managers, etc.
  - A little about Usenet discussion groups.

- Reminder: Homework 5 due today.