Slide 1

Administrivia

- Reminder: Homework 5 due Monday.

Slide 2

Minute Essay From Last Lecture

- Most people reported finding at least a few uses.
- Specific mentions were of `vim` features (it's so much less painful if you know more of them), `awk`(!), scripting to facilitate testing (for a project for the compilers course), and various scripts to do things of personal interest/use.
- One person did mention that it's not clear that the time invested in developing scripts really pays off in terms of time saved doing repetitive tasks, but it does feel more productive?

Slide 3

Text Editors and Regular Expressions, Once(?) More

- Updated versions of script to compute factorial on “sample programs” page, one using a regular expression to check whether the argument is an integer and another using another method.
- `vim`’s macro feature isn’t always a help (as shown last time, alas) but sometimes is. A better “use case” for me: Add to my raw-HTML index file for the “sample programs” page entries for recently-created examples.
- Last time I mentioned a script to summarize information from a `procmail` log. Try that again?
- As one more example, I have a script I run from inside `vim` on those text files containing homework grades and comments to compute score based on perfect score and deductions/additions.

Slide 4

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 parts that are affected.
- Idea behind `make` — have computer figure out what needs to be recompiled and issue right commands to recompile it.

Slide 5

Makefiles

- First step in using `make` is to set up "makefile" describing how 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`.

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

Slide 6

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 7

Defining Rules

- Define dependencies for a rule by giving, for each “target”, list of files it depends on.
- Also give the list of commands to be used to recreate 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 8

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`.

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)
```

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`.

Minute Essay

- None really — just sign in, unless you have questions?