CSC161 2011S *Imperative Problem Solving*

Class 34: Make, Revisited

Held: Tuesday, 5 April 2011

Summary: We consider more details of `make`, a useful tool for automating lots of stuff.

Related Pages:

- EBoard.
- Reading: *Managing Projects with GNU Make*, Chapters 1 and 2 .

Notes:

- Responses to your questions on the make reading have been posted. We will also talk about them in class.
- Missing questions from AA, DG, MH, JJ, SR, MT.
- Why do I give you complex and hard to parse readings? Because I think you need to develop the skills to read complex technical things (rather than pre-processed pablum), even if you won't be CS majors.
- Reading for tomorrow: K&R 5.5, B3.
- Reading for Friday: Debugging with DDD.
- Update on Kington presentation.
- EC for Wednesday's M.C. Escher talk [academic]
- EC for Thursday's nonvo on Iowa Water (or other part of that symposium) [academic]
- EC for Friday's presentation of *Sita Sings the Blues* [academic]
- EC for today's talks on Alt-Break.
- EC for any activity in the animation festival, including Tiny Circus [peer]
- EC for either part of the baseball doubleheader against Knox [peer]

Overview:

- About Make.
- File Suffixes.
- Your Questions.
- A Collaborative Makefile.

Make

- Purpose: Make it easier to build projects, particularly complex multi-part projects.
- We'll look at two sample projects:
  - The course Web site (or portions thereof)
  - A recent program you worked on (or my version thereof)
- Model: A collection of "targets"

    - Each target is something that we might want to build (or a placeholder to help us build stuff)
    - There are instructions for building each target.
- Typical targets:
    - `default`: The default thing or things to build (e.g., the application or library)
    - `test` or `check`: Instructions for testing the main thing. (Generally predicated on building `default` first.)
    - `install`: Install the things we've just built.
    - `clean`: Remove intermediate files (such as .o files).
    - `distclean`: Remove everything but the source files.
    - `package`: Put everything together into a tarball.
- Two kinds of targets
    - File targets: Things we actually build
    - Command targets: A shorthand for something we want to do, even if it might not build any files. (clean and test are examples) These are also called phony targets.
- Variables: Making it easier. Four kinds:
    - Your variables
    - Standard variables you set
    - Variables from the standard rules
    - Automatic variables
- Your variables

```
NAME = VALUES
```

- Standard variables you set
    - CFLAGS
    - LDFLAGS
    - LDLIBS
- Variables from the standard rules
    - `$(CC)`
    - `$(COMPILE.c)`
    - ...
- Automatic variables
    - `$@` - The target
    - `$<` - The first prereq
    - `$?` - Newer prereqs
    - `$^` - All prereqs
    - `$*` - The stem of the target

File Suffixes

- In Unix/Linux, as in many operating systems, we give a hint to the contents of a file by adding a suffix.
- You already know some suffixes: `.c`, `.o`, and `.h`.
- Some other useful ones

    - `.a` - an archive/library
    - `.s` - assembly code
    - `.sh` - a shell script (collection of shell commands)
- Some other ones mentioned in the text but that you're unlikely to use
    - `.l` - Lex/Flex source code
    - `.so` - a shared object (like a DLL)

Your Questions

- See the separate reading page.
- Some possible demonstrations:
    - Libraries
    - VPATH and vpath
    - CPPFLAGS
    - .PRECIOUS
    - .PHONY

Exercise

- We'll work together to build a nice Makefile for my files from problem 3 on the exam.
