CSC 195.01 2013S *Technologies for Mediascripting*

Class 03: Automation with Make

Held: Thursday, 7 February 2013

Summary: We consider more details of `make`, a useful tool for automating steps in building programs on *nix systems.

Related Pages:

- EBoard.

Notes:

- I encourage you to go to today's Thursday extra, which is by some of my research students.
- This week's homework: Find some task you do on our Linux system and write a Makefile to support it.
- We'll start today's class with a short discussion of your experiences with git.

Overview:

- An introduction to Make
- Example 1: Some fun with text
- Good Make practice: Standard targets
- Example 2: A standard C project
- Using variables

An introduction to Make

- Purpose: Make it easier to build projects, particularly complex multi-part projects.
- We'll look at two sample projects:
  - Various forms of text files
  - A simple set of C programs.
- Model: A collection of "targets"
  - A target is often a file that we want to build. For example, we might want to build "book.pdf".
  - A target can also be a placeholder name (e.g., "pdf" for "all the PDF files"). These are often called "phony targets"
  - To build a target, we follow a sequence of instructions.
  - Targets often depend upon other files.
- At the most basic level, a Makefile contains a collection of targets and instructions for building the targets.
- Format:

```
target: required-file-1 required-file-2 ...
        instruction-1
        instruction-2
        ...
```

  - Note that the instructions *must* be preceded by a tab character (and not the corresponding number of spaces)
- All of the instructions get stored in a file named `Makefile`
  - There are ways to give the file other names, but I'm not going to teach them to you.

Example 1: Some Fun with Text

- Documents can appear in many forms.
- In this activity, we'll consider "plain" text (more or less), HTML, Postscript, and PDF.
- The MarkDown format provides a relatively nice way to convert plain text to formatted HTML.
- html2ps is useful for converting HTML to Postscript, which is a nice form for printing.
- ps2pdf converts ps files to pdf, which can then be viewed onscreen.
- See Examples/Make/Text for more details.

Good Make Practice: Standard Targets

- By convention, Makefiles include a variety of targets, so that people who get your project can just type `make` *`standard-target`* to acheive a particular goal.
- Typical targets (almost all are placeholders):
  - `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.
- I would recommend that you support default, test, and clean in all of your Makefiles.
- When you are packaging code for others, install, distclean, and package are particularly useful.

Example 2: C Programming

Warning! This example may use more C than you know. Let me know when you have questions.

- Typically, C programs are broken up into many separate files.
- Typically, different (but overlapping) subsets of those files are used to make different applications.
- Here's a simple (but fairly standard) project I've set up.
  - `srmath.c` is a math library I've designed and want to use in various projects. (Right now, it only incluldes a gcd function.)
  - `srmath.h` is the header file that the C compiler uses for type checking and other similar purposes.

    - `srtest.c` is my testing library.
    - `srtest.h` is the header file for my testing library.
    - `gcdtest.c` is a test program for the `gcd` function.
    - `gcd.c` is a user interface to the gcd function (that is, a program you can run from the command line).
- So, how do I put things together?
- To build my test, I need to
    - Compile srmath.c to srmath.o.
    - Compile srtest.c to srtest.o.
    - Compile gcdtest.c to gcdtest.o.
    - Link srmath.o srtest.o and gcdtest.o to gcdtest
- Building my gcd application is similar. In that case, I need to
    - Compile srmath.c to srmath.o.
    - Compile srtest.c to srtest.o.
    - Compile gcd.c to gcd.o.
    - Link srmath.o srtest.o and gcd.o to gcd
- Some parts of this are automated by Make, even if I don't type anything.
    - E.g., I can type `make srmath.o` without creating a rule.
- Other parts require instructions
    - E.g., I need a rule for "Link srmath.o srtest.o and gctest.o"
- Here's a question: Can Make tell what work needs to be redone if I change a file?

Simplifying Makefiles with Variables

- As you've already found when programming, variables tend ot make your life easier.
- Make has four kinds of variables (more or less)
    - Your variables
    - Standard variables you set
    - Variables from the standard rules
    - Automatic variables
- You access every variable with $*var* or $(*var*)
- You set most variables with *var = VALUE*
- Your variables
    - You get to choose the name and purpose
- Standard variables
    - Used by Make in common rules, but also by some other programs
    - Some examples
        - `CFLAGS`
        - `LDFLAGS`
        - `LDLIBS`
- Variables from the standard rules
    - Also come from the common rules
    - Some examples

        - `$(CC)`
        - `$(COMPILE.c)`
        - ...
- Automatic variables
    - Computed by Make from the rule
    - End up being useful in writing more concise or more general rules
    - Some of my favorites
        - `$@` - The target
        - `$<` - The first prereq
        - `$?` - Newer prereqs
        - `$^` - All prereqs
        - `$*` - The stem of the target
- We'll work on updating our sample Makefiles to use variables.

Copyright © 2012 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 3.0 License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/3.0/` or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.