15 - Build Systems, Git Merging & Working Across Branches

CS 2043: Unix Tools and Scripting, Spring 2016 [1]

Stephen McDowell

March 2nd, 2016

Cornell University

Table of contents

Some Logistics

- Updates to the demos and some backlogged lectures are up.
- Python goodies, why I had you install `ipdb` (and Python 3)
- The Week of the 18th proposals (purely supplemental):
  - Monday, March 14th: how to install Linux natively.
  - Wednesday, March 16th: in-depth build-systems, examples on compiling from source and when you may need to do it.
  - Friday, March 18th: tournament? Hosted by not me (out of town).
  - Suggestions welcome if you would rather see something else.
  - Alternate possibility: filesystems, automounting, management, growing / shrinking volumes.
- HW2 due tonight...

Build Systems

What for?

- Build systems are there to make your life easy. It would be entirely infeasible to require an individual user to compile everything on their own without guidance.
- With good build systems comes the implicit necessity for good documentation!
    - A `README` at the very least, preferably an `INSTALL` file with further guidance, listing of required packages, platform notes (if applicable), etc.
- The core concept: automate as much as possible.
    - If for whatever reason you have to compile the source (your own project, need alternative functionality), you will *need* to know how to use these tools.

Build Systems in the Wild

- You will likely encounter the following kinds of build systems:
  - A `Make` project (just includes a `Makefile`).
  - A `CMake` project (includes a `CMakeLists.txt` file).
  - An auto-tools project (usually of the form `setup.sh`).
  - A Python build (`python setup.py install`).
- Each have their quirks and benefits.
  - You may have to create your own.
  - Or you may be able to get away with just knowing how to execute them.
  - It very much depends on the situation.

Make

- Manage compilation of programs written in languages like `C/C++`.
- Used to automatically update any set of files that depend on another set of files.
- The `Makefile` (capital `M`) is the proper name:
    - If there exists a `Makefile` in the current directory, just execute `make`.
        - ...assuming it was written correctly...
    - Can execute `make -f <filename>` if named something else.
- The `Makefile` describes how files depend on each other, and how to update out-of-date files.
- Makes use of patterns, rules, and variables to eliminate redundancy.
- Uses macros and control operation.

A Sample Makefile

```
myapp: file1.o file2.o
    gcc -o myapp file1.o file2.o
file1.o: file1.c macros.h
    gcc -c file1.c
file2.o: file2.c macros.h
    gcc -c file2.c
```

- Describes the dependencies of `myapp`: the compiled `file1` and `file2` object files.
- These dependencies are recursively defined in the subsequent `file1.o` and `file2.o` targets.
- Both of these targets depend on `macros.h`.
- You can define as many targets as you need.

Make Specifics

- Properly defined? `.PHONY`, `all`, `clean`
- Must use `tab` characters. ALWAYS. ewwwww....
- Automatic generation magic.
- Lecture slides `Makefile`.
- The syntax is pretty crazy.
- `make` followed by `sudo make install`

CMake

- Configure Make.
- Cross-platform if done right.
- Example `nori`.
- CCMake: Configure CMake. LOL.
- Creates build systems for you.
    - General idea (on Unix systems):

```
>>> mkdir build
>>> cd build
>>> cmake ..
>>> ccmake ..
>>> make
```

Autotools

Basically you just run `setup.sh`. If it fails, the standard is to tell you exactly why, e.g. point you to files that you need or libraries you need to install.

Python Setup

- Generally: `python setup.py install`
- You may need to put a `sudo` in front of that.

Packaging your Packages

- Make an `rpm`:

  http://www.thegeekstuff.com/2015/02/rpm-build-package-example/

- Make a `ppa`:

  http://askubuntu.com/questions/71510/how-do-i-create-a-ppa

Merging Like a Boss

Lets do it

```
# http://www.rosipov.com/blog/use-vimdiff-as-git-mergetool/
git config merge.tool vimdiff
git config merge.conflictstyle diff3

# http://stackoverflow.com/a/1251696/3814202
git config --global mergetool.keepBackup false
```

Working Across Branches

What do you take from where?

- `git pull origin <branch>`
- `git checkout <branch> -- file`
- `git ls-tree`
- `get crazy with it`

References I

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.
Previous cornell cs 2043 course slides.