Project Makefile: Slides

Generic Python Project Makefile

Configuration Management Working Group

October 25, 2016

Makefile for Python Web Development & Related Projects

Who

- **Alex Clark**
  - **Python Web Developer (for hire, forever)**
  - Pillow (Python Imaging Library fork) author
  - Partner/President & Executive Director (ACLARK.NET, LLC/DC Python)
  - Systems Administrator (NIH)
  - (former) Network Engineer (BBN)
  - Would-be Computer Science PhD candidate
  - "Hacker" (I <3 UNIX & command line)

(Not a GNU Make expert)

What

- **Makefile** for Python Web Development & related projects.

  - Django (startapp, startproject, etc.)

  - Python packaging (flake8, yapf, etc.)

  - Heroku (debug, push, etc.)

Where

- https://github.com/aclark4life/**project-makefile**

When

- Initial commit: **January 12, 2016**
- https://github.com/aclark4life/project-makefile/commit/01a85c746d4a06058fcb4749e85186325449f34b
- Originally "django-project"

Why

- Laziness e.g.
  - `$ make world`
- Familiarity e.g.
  - `$ ./configure; make; make install`
- Simplicity e.g.
  - `$ cp ~/Developer/project-makefile <DIR>`

Car Analogy

If <YOUR_FAV_CM_SOFT> is a sports car...

```
              ____----------- _____
\~~~~~~~~~~/~_--~~~------~~~~~     \
 `---`\  _-~      |                   \
   _-~  <_         |                     \[]
 / ___     ~~--[""] |      ________-------'_
> /~` \    |-.   `\~~.~~~~~                _ ~ - _
 ~|  ||\%  |       |    ~  ._                ~ _   ~ ._
   `_//|_%  \      |          ~  .              ~-_   /\
          `--__     |    _-____  /\               ~-_ \/.
               ~--_ /  ,/ -~-_ \ \/          _______---~/
                   ~~-/._<   \ \`~~~~~~~~~~~~~     ##--~/
                         \    ) |`------##---~~~~-~  ) )
                          ~-_/_/                  ~~ ~~
```

E.g. Ansible, Chef, Puppet

This Makefile is. . .

A Volkswagen Beetle

(the old one)

```
                                 _________
         This             _.--""'-----,   `"--.._
       Makefile        .-''   _/_      ; .'"----,`-,
                     .'      :___:     ; :      ;;`.`.
                    .      _.- _.-    .' :      ::  `..
                 __;..----------------' :: ___  ::   ;;
            .--"". '           ___.....`:=(___)-' :--'`.
          .'   .'         .--''__       :       ==:    ;
      .--/    /        .'.''     ``-,   :         :   '`-.
   ."', :    /       .'-`\\       .--.\ :         :  ,   _\
  ;   ; |   ;       /:'  ;;      /__  \\:         :  :  /_\\
  |\_/  |   |      / \__//      /"--\\ \:         :  : ;|`\|
  : "  /\__/\____//   """      /     \\ :         :  : :|'||
["""""""""--------........._  /      || ;      __.:--' :|//|
 "------....______         ].'|      // |--"""'__...-'`\ \//
   `| GNUMAKE |__;_...--'": :  \    //  |---"""      \__\_/
     """""""""'            \ \  \_.//  /
       `---'                \ \_     _'
                             `--`---'  dp
```

Context is Key

Possible Alternatives, Dependent on Context

- **Ansible, Chef, Puppet**
- Buildout
  - `[buildout] extends = http://your-project`
- Shell
  - `alias do="echo do stuff"`
- Fabric
  - `def do_stuff(): print("do stuff")`
- Grunt
- Rake

Questions

Ask anytime

Overview of GNU Make

- "The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them... Our examples show C programs, since they are most common, but **you can use make with any programming language** whose compiler can be run with a shell command. Indeed, **make is not limited to programs**. You can use it to **describe any task where some files must be updated automatically** from others whenever the others change." (https://www.gnu.org/software/make/manual/html_node/Overview.html#Overview)

Overview of GNU Make

- "Certain standard ways of remaking target files are used very often. For example, one customary way to **make an object file is from a C source file** using the C compiler, cc." (https://www.gnu.org/software/make/manual/html_node/Implicit-Rules.html#Implicit-Rules)

GNUMakefile
Makefile
makefile

```
all: average counter list roach
average:
    gcc average.c -o average
counter:
    gcc counter.c -o counter
list:
    gcc list.c -o list
roach:
    gcc roach.c -o roach
trapezoid:
    gcc trapezoid.c -o trapezoid
clean:
    rm average counter list roach
```

A Makefile

Just
Type
$ make

```
all: average counter list roach
average:
    gcc average.c -o average
counter:
    gcc counter.c -o counter
list:
    gcc list.c -o list
roach:
    gcc roach.c -o roach
trapezoid:
    gcc trapezoid.c -o trapezoid
clean:
    rm average counter list roach
```

Without Implicit Rule Usage

```
objects = average.o
all: average counter list roach
average: $(objects)
    gcc -o average $(objects)
counter:
    gcc counter.c -o counter
list:
    gcc list.c -o list
roach:
    gcc roach.c -o roach
trapezoid:
    gcc trapezoid.c -o trapezoid
clean:
    rm average counter list roach
```

With Implicit Rule Usage

See what I did there?

Replaced explicit file (.c) compilation with implicit object (.o) compilation.

Implicit Behavior is Awesome!

(Except in Python)

Terminology
"Makefile Moment #0"

- "A ***variable*** is a name defined in a makefile to represent a string of text, called the variable's *value*. These values are substituted by explicit request into targets, prerequisites, recipes, and other parts of the makefile. (In some other versions of make, variables are called *macros*.) (https://www.gnu.org/software/make/manual/html_node/Using-Variables.html)

Installation

- `mkdir <YOUR-PROJECT>`
- `cd <YOUR-PROJECT>`
- `curl -O https://raw.githubusercontent.com\ /aclark4life/project-makefile/master/Makefile`

Usage

```
$ make help
Usage: make [TARGET]
Available targets:

    - ablog-build
    - ablog-init
    - ablog-serve
...
```

Terminology

“Makefile Moment #1”

- A ***target*** is usually the name of a file that is generated by a program; examples of targets are executable or object files. A target can also be the **name of an action to carry out**, such as ‘clean’ (see Phony Targets). (https://www.gnu.org/software/make/manual/make.html#Rule-Introduction)

Example #1

```
$ make help
```

Target

```
help:
    @echo "Usage:...
```

```
help:
    @echo "Usage: make [TARGET]\nAvailable targets:\n"

    @$(MAKE) -pRrq -f $(lastword $(MAKEFILE_LIST)) : 2>/dev/null | awk -v RS= -F:\
        '/^# File/,/^# Finished Make data base/ {if ($$1 !~ "^[#.]") {print $$1}}'\
        | sort | egrep -v -e '^[^[:alnum:]]' -e '^$@$$' | xargs | tr ' ' '\n' | awk\
        '{print "    - "$$0}' | less  # http://stackoverflow.com/a/26339924

    @echo "\n"
```

Terminology

“Makefile Moment #2”

- "The ***goals*** **are the targets** that make should strive ultimately to update. Other targets are updated as well if they appear as prerequisites of goals, or prerequisites of prerequisites of goals, etc.
  By default, the goal is the first target in the makefile (not counting targets that start with a period). Therefore, makefiles are usually written so that the first target is for compiling the entire program or programs they describe… You can manage the selection of the default goal from within your makefile using the .DEFAULT_GOAL variable " (https://www.gnu.org/software/make/manual/html_node/Goals.html)

```
targets : prerequisites
        recipe
        ...
```

(https://www.gnu.org/software/make/manual/html_node/Rule-Syntax.html#Rule-Syntax)

Terminology

Makefile Moment #3

Example #2

```
$ make
```

Target

```
.DEFAULT_GOAL=git-commit-auto-push
```

...

git-commit-auto-push: git-commit-auto git-push

```
# Git
MESSAGE="Update"
...
commit-auto: git-commit-auto  # Alias
commit-edit: git-commit-edit  # Alias
git-commit: git-commit-auto  # Alias
git-commit-auto-push: git-commit-auto git-push
...
git-commit-auto:
    git commit -a -m $(MESSAGE)
git-commit-edit:
    git commit -a
git-push:
    git push
```

Example #3

```
$ make ablog-init
```

Target

ablog-init:
    bin/ablog start

A Bit

About…

Python

Python

- virtualenv
- pip
- requirements.txt
- source bin/activate
- bin/python

Example #4

```
$ make ablog
```

Target

```
ablog: ablog-clean \
         ablog-install\
         ablog-init \
         ablog-build \
         ablog-serve
```

Example #5

```
$ make django
```

Target

```
django: django-clean \
        django-install\
        django-init \
        django-migrate\
        django-su\
        django-serve
```

Terminology

"Makefile Moment #4"

- "A ***rule*** appears in the makefile and says when and how to remake certain files, called the rule's *targets* (most often only one per rule). It lists the other files that are the *prerequisites* of the target, and the *recipe* to use to create or update the target." (https://www.gnu.org/software/make/manual/make.html#Rules)

Example #6

```
$ make sphinx
```

Target

```
sphinx: sphinx-clean \
    sphinx-install \
    sphinx-init \
    sphinx-build \
    sphinx-serve
```

A Bit

About…

JavaScript

JavaScript

- NPM
- webpack
- Grunt

Example #7

```
$ make grunt
```

Target

```
grunt: grunt-init \
       grunt-serve
```

Terminology "Makefile Moment #5"

- Ignore errors in a recipe with "-"
  - -ls one
  - -ls two
  - ls three

Example #8

```
$ make npm
```

Target

```
npm: npm-init \
        npm-install
```

A Bit

About...

Apps/Utils

Apps/Utils

- ABlog
- Plone
- Sphinx
- Vagrant

Example #9

```
$ make plone
```

Target

```
plone: plone-install \
        plone-init\
        plone-serve
```

Example #10

```
$ make vagrant
```

Target

```
vagrant: vagrant-init
```

Example #12

```
$ make django
```

Target

```
django: django-clean\
       django-install\
       django-init \
       django-migrate\
       django-su \
       django-serve
```

```
$ make lint
```

```
lint: python-lint
python-lint: python-flake\
        python-yapf\
        python-wc
```

Pros

- Easy to install and use
- Unified usage of various technologies

Cons

- Arguably wrong tool for the job
- Static PROJECT/APP definition

Todo

- Support customization
  - Include other makefiles
  - Redefine targets
- Support additional programming languages, apps, utils, etc.

Recap

- GNU Make
  - Targets, Recipes, Rules, Prerequisites
- Python
  - pip, virtualenv, requirements.txt
- JavaScript

  - Grunt, Node Package Manager (NPM)
- Apps & Utils
  - ABlog, Plone, Sphinx, Vagrant

The End

- https://github.com/aclark4life/project-makefile
- http://slides.com/aclark/project-makefile

Credits/Thanks

- Jeff @ CMWG, attendees, reviewers
- https://github.com/hakimel/reveal.js
  - https://slides.com
- http://www.ascii-code.com/ascii-art/vehicles/cars.php