Pimp my Makefile

Michel Casabianca
casa@sweetohm.net

As you don't want to repeat yourself, it's a good practice to put all tasks that you might run twice somewhere in your project. A Makefile is a good place to do so and it is also an executable documentation: instead of documenting the build process, you should write it in a *build* target of your Makefile.

Make might not be the best build tool, but it is almost everywhere, at least installed or a command away in all Linux distributions. But it is far from perfect: for instance there is no integrated help or option to list available targets in order to perform Bash completion.

Help on Targets

Let's consider following typical Makefile:

```
BUILD_DIR=build

clean: # Clean generated files and test cache
	@rm -rf $(BUILD_DIR)
	@go clean -testcache

fmt: # Format Go source code
	@go fmt ./...

test: clean # Run unit tests
	@go test -cover ./...

.PHONY: build
build: clean # Build binary
	@mkdir -p $(BUILD_DIR)
	@go build -ldflags "-s -f" -o $(BUILD_DIR)/hello .
```

Make doesn't provide any option to list available targets along with documentation extracted from comments. Let's do it:

```
BUILD_DIR=build

help: # Print help on Makefile
	@grep '^[^.#]\+:\s\+.*#' Makefile | \
	sed "s/\(.\+\):\s*\(.*\) #\s*\(.*\)/`printf "\033[93m"`\1`printf "\033[0m"`
	expand -t20

clean: # Clean generated files and test cache
	@rm -rf $(BUILD_DIR)
	@go clean -testcache
```

```
fmt: # Format Go source code
	@go fmt ./...

test: clean # Run unit tests
	@go test -cover ./...

.PHONY: build
build: clean # Build binary
	@mkdir -p $(BUILD_DIR)
	@go build -ldflags "-s -f" -o $(BUILD_DIR)/hello .
```

Now you can get help on targets typing:

```
$ make help
help       Print help on Makefile []
clean      Clean generated files and test cache []
fmt        Format Go source code []
test       Run unit tests [clean]
build      Build binary [clean]
```

Target *help* parses Makefile with a regexp to extract target names, descriptions and dependencies to pretty print them on terminal. As this target is the first in the Makefile, this is the default one and you can get help typing `make`.

Bash Completion on Targets

Some distributions provide a package to add Bash completion on Make targets, some don't. If you don't have completion while typing `make [TAB]`, you can add it by sourcing following file (in your *~/.bashrc* file for instance):

```
# /etc/profile.d/make

complete -W "\`grep -oEs '^[a-zA-Z0-9_-]+:([^=]|$)' ?akefile | sed 's/[^a-zA-Z0-9_.
```

With example build file, completion would print:

```
$ make [TAB]
build  clean  fmt    help   test
$ make t[TAB]est
```

This is very handy with big Makefile with many targets.

Makefile Inclusion

It is possible to include other Makefile with `include` directive. For instance, with Makefile *help.mk* in same directory:

```
help: # Print help on Makefile
        @grep '^[^.#]\+:\s\+.*#' Makefile | \
        sed "s/\(.\+\):\s*\(.*\) #\s*\(.*\)/`printf "\033[93m"`\1`printf "\033[0m"`
        expand -t20
```

You could import it in your main Makefile as follows:

```
include help.mk

BUILD_DIR=build

clean: # Clean generated files and test cache
        @rm -rf $(BUILD_DIR)
        @go clean -testcache

fmt: # Format Go source code
        @go fmt ./...

test: clean # Run unit tests
        @go test -cover ./...

.PHONY: build
build: clean # Build binary
        @mkdir -p $(BUILD_DIR)
        @go build -ldflags "-s -f" -o $(BUILD_DIR)/hello .
```

This will include *help.mk* with its target *help*. But as target help is no longer in the main Makefile, it will not appear anymore while printing help:

```
$ make help
clean      Clean generated files and test cache []
fmt        Format Go source code []
test       Run unit tests [clean]
build      Build binary [clean]
```

Likewise, Bash completion will not include target help for the same reason. To enable help and completion with included Makefiles, this would require more work to parse them and take included targets into account.

Make Tools

Make Tools were made to solve these inclusion issues. There are two of these tools:

Make Help

This tool scans current directory to find makefile, parses it to extract targets information and included makefiles to process recursively. Thus to print help in previous example, you would type:

```
$ make-help
build Build binary [clean]
clean Clean generated files and test cache
fmt   Format Go source code
help  Print help on Makefile
test  Run unit tests [clean]
```

We notice that targets are sorted and *help* target is included in printed help.

You might include this help target with following definition in a makefile:

```
.PHONY: help
help: # Print help on Makefile
	@make-help
```

Make Targets

This tool lists targets of makefile in current directory and all included ones recursively. With previous example:

```
$ make-targets
build clean fmt help test
```

Thus, to perform bash completion, you should source:

```
complete -W "\`make-targets\`" make
```

Known Bugs

These tools behave as make does:

- Included files are relative to current directory, not to makefile including them.
- There is no infinite loop detection for inclusions.

This tool is open source, feel free to contribute to improve it.

Enjoy!