Pimp my Makefile

Michel Casabianca
casa@sweetohm.net

Un bon moyen pour ne pas se répéter est d'écrire les tâches que vous pourriez réaliser plusieurs fois quelque part dans votre projet. Un *makefile* est le bon endroit pour le faire, et c'est aussi une documentation exécutable : au lieu de documenter le processus de build, vous devriez écrire une cible *build* dans votre *makefile*.

Make n'est probablement pas le meilleur outil de build, mais il est partout, présent dans toutes les distributions Linux ou installé d'une simple commande. Mais il est loin d'être parfait : il n'y a pas d'aide intégrée ni de moyen de lister les cibles afin d'effectuer la complétion avec Bash.

Aide sur les cibles

Considérons le makefile typique suivant :

```
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 ne fournit pas d'option pour lister les cibles avec leur documentation extraite des commentaires. Faisons le :

```
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 .
```

Maintenant vous pouvez obtenir de l'aide en tapant :

```
$ 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]
```

La cible *help* parse le makefile à l'aide d'une expression rationnelle pour en extraire les noms, la description et les dépendances des cibles pour les afficher dans le terminal. Comme cette cible est la première du makefile, c'est aussi la cible par défaut que l'on peut appeler en tapant simplement `make`.

Complétion Bash pour les cibles

Certaines distributions proposent un package pour ajouter la complétion Bash sur les cibles de Make, d'autres non. Si vous n'avez pas de complétion en tapant `make [TAB]`, vous pouvez l'ajouter en sourçant le fichier suivant (dans votre fichier *~/.bashrc* par exemple) :

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

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

Avec le makefile de l'exemple, la complétion affichera :

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

Ceci est bien pratique pour de grands makefiles avec de nombreuses cibles.

Inclusion de makefiles

Il est possible d'inclure d'autres makefiles avec la directive `include`. Par exemple, avec le

makefile *help.mk* dans le même répertoire :

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

Vous pouvez l'importer dans votre makefile principal comme suit :

```
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 .
```

Ceci va inclure *help.mk* avec sa cible *help*. Mais comme cette cible n'est plus dans le makefile principal, elle n'apparaît plus dans l'aide :

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

De même, la complétion Bash n'inclura pas la cible *help* pour la même raison. Pour permettre l'aide et la complétion sur les makefiles inclus, cela demande plus de travail pour les parser et prendre en compte les cibles incluses.

Make Tools

Make Tools ont été développés pour régler ces problèmes d'inclusion. Il y a deux outils :

Make Help

Cet outil recherche le makefile dans le répertoire courant, le parse pour extraire les informations sur les cibles et les makefiles inclus de manière récursive. Pour afficher l'aide de l'exemple précédent,

vous taperez :

```
$ 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]
```

Nous noterons que les cibles sont triées par ordre alphabétique et la cible *help* est incluse.

Vous pouvez inclure cette cible *help* avec la définition suivante dans le makefile :

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

Make Targets

Cet outil liste les cibles du makefile dans le répertoire courant et de tous les makefiles inclus. Avec l'exemple précédent :

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

Donc pour réaliser la complétion avec Bash, vous pouvez sourcer :

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

Bugs connus

Ces outils se comportant comme *make* :

- Les fichiers inclus sont relatifs au répertoire courant et non au répertoire du makefile qui les inclut.
- Il n'y a pas de détection des boucles infinies pour les inclusions.

Cet outil est Open Source, n'hésitez pas y contribuer pour l'améliorer.

Enjoy!