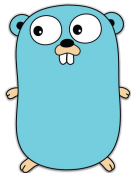
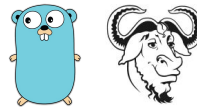


Build Golang projects properly with Makefiles

Raül Pérez - [@repejota](#) - 25/11/2015





Who am I?

Hi! My name is Raúl Pérez

- Lead Software Engineer at Carrenza Ltd. <http://www.carrenza.com>
- Living between Barcelona & London.
- Working on devops stuff, but I'm still more a Dev than an Op.
- Proud to be a Gopher, but I'm not a fanboy. I write code on multiple languages.

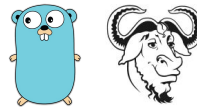
Twitter: <https://twitter.com/repejota>

Github: <https://github.com/repejota>

LinkedIn: <https://linkedin.com/in/repejota>

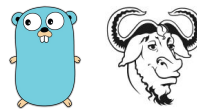
A close-up, dimly lit photograph of a person's hands carving a chocolate beetle. The person is using a small, sharp carving tool with a yellow handle. The beetle is being carved on a dark, possibly black, surface. There are many small, brown chocolate shavings scattered around the beetle. In the background, there is a dark, textured surface, possibly a piece of wood or a stone, and a small, round, dark object, possibly a piece of chocolate or a small bowl. The overall atmosphere is focused and artistic.

Why Makefiles?



Why using Makefiles?

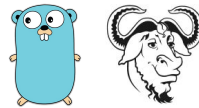
- Build != Compile
 - Building a project means more than just compile your code.
 - Static analysis, Linters, Installation, configuration, packaging, etc ...
- Same tool on each phase of the project.
 - Makefiles act as the glue to go tools (build, install, clean)
 - But can use also third party tools (go lint, go vet, gb, etc ...)
- Language agnostic.
 - Probably your project does not rely on just one language.
 - Generate configuration files, man pages, fixture data, etc ...
- Available on multiple platforms.
 - Because users use whatever they want.
 - Cross compilation.



Why using Makefiles? (II)

Three goals are necessary to accomplish when you **properly** want to build your project:

1. Easy & portable compilation, installation and cleanup.
2. Make trivial to use compile time variables.
3. Reproducible builds.

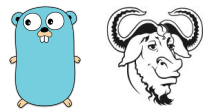


Example code to build.

```
package main

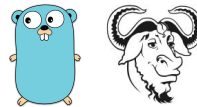
import "fmt"

func main() {
    // Probably the most awesome piece of code you've ever seen.
    fmt.Println("foo bar")
}
```



Example Makefile

```
default:
    go build
```

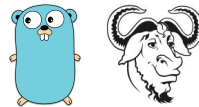


Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ ls -al
total 4560
drwxr-xr-x  5 raul  staff    170 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff    306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff     19 25 Nov 03:11 Makefile
-rwxr-xr-x  1 raul  staff 2324096 25 Nov 05:08 gomake
-rw-r--r--  1 raul  staff    132 25 Nov 05:07 main.go
$ ./gomake
foo bar
$
```


A top-down view of various vintage tools and items arranged on a dark, vertically-grained wooden surface. The items include two axes with wooden handles and metal heads, a claw hammer, a mallet, a pair of large pliers, a pair of tan leather work gloves, a metal mug, a small metal bell, a utility knife, a small metal container with a logo, and a curved metal tool. The text "Easy & portable compilation, installation and cleanup" is overlaid in white, bold, sans-serif font in the center of the image.

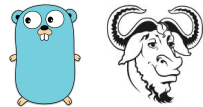
**Easy & portable compilation,
installation and cleanup**



Easy compilation, installation and cleanup (I).

Compilation: Just type “make”

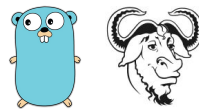
- Laziness! Less keystrokes than “go build <package>”
- You can also build multiple projects with a single Makefile.
 - A Makefile could call to another Makefile.
- Want to use more sophisticated build tool with your code?
 - All scripts and data to build a project belongs to the project itself.
 - Just change the Makefile not your continuous integration process.



Easy compilation, installation and cleanup (II).

Installation: Just type “make install”

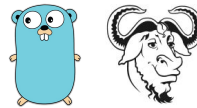
- Most of the times “go install <package>” is not enough, there is probably extra steps to do after creating the binary.
- Generate configuration files, man pages, fixture data, etc ...
- Just change the Makefile not your deployment process.



Easy compilation, installation and cleanup (III).

Cleanup: Just type “make clean”

- Cleaning your project means more than just deleting your old binary.

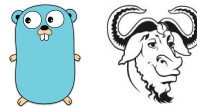


Improved Makefile

```
# Default target: builds the project
build:
    go build

# Installs our project: copies binaries
install:
    go install

# Cleans our project: deletes binaries
clean:
    go clean
```

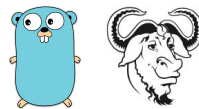


Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ ls -al
total 4560
drwxr-xr-x  5 raul  staff    170 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff    306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff     19 25 Nov 03:11 Makefile
-rwxr-xr-x  1 raul  staff 2324096 25 Nov 05:08 gomake
-rw-r--r--  1 raul  staff    132 25 Nov 05:07 main.go
$ ./gomake
foo bar
$ make clean
go clean
$
```

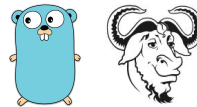


**Make trivial to use compile
time variables**



Make trivial to use compile time variables

- Adding a VERSION number.
- Adding the Build Time.



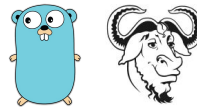
Improved source code.

```
package main

import (
    "fmt"
)

// Variables to identify the build
var (
    Version = "1.0.0"
    Build = "2015-11-25T00:23:32+0100"
)

func main() {
    fmt.Println("Version: ", Version)
    fmt.Println("Build Time: ", Build)
}
```



Improved Makefile

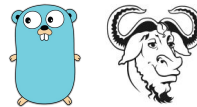
```
# This how we want to name the binary output
BINARY=gomake

# Builds the project
build:
    go build -o ${BINARY}

# Installs our project: copies binaries
install:
    go install

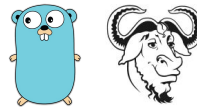
# Cleans our project: deletes binaries
clean:
    if [ -f ${BINARY} ] ; then rm ${BINARY} ; fi

.PHONY: clean install
```



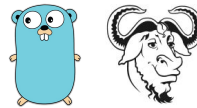
Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ ls -al
total 4560
drwxr-xr-x  5 raul  staff    170 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff    306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff     19 25 Nov 03:11 Makefile
-rwxr-xr-x  1 raul  staff 2324096 25 Nov 05:08 gomake
-rw-r--r--  1 raul  staff    132 25 Nov 05:07 main.go
$ ./gomake
Version:  1.0.0
Build Time:  2015-11-25T00:23:32+0100
$ make clean
if [ -f gomake ] ; then rm gomake ; fi
$
```



Make trivial to use compile time variables (II)

- Adding a VERSION number.
 - Editing the code to bump the version is painful.
 - Your are going to forget it, commit, and your git history is going to be broken.
- Adding a Build Time.
 - It happens the same than with the VERSION.
 - Which format of timestamp to use? It is a weird string!
 - And obviously you need a clock to check the exact time before! :P



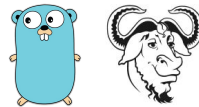
Improved source code.

```
package main

import (
    "fmt"
)

// Variables to identify the build
var (
    Version string
    Build string
)

func main() {
    fmt.Println("Version: ", Version)
    fmt.Println("Build Time: ", Build)
}
```



Improved Makefile

```
# This how we want to name the binary output
BINARY=gomake

# These are the values we want to pass for VERSION and BUILD
VERSION=1.0.0
BUILD=`date +%FT%T%z`

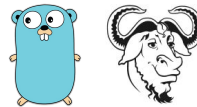
# Setup the -ldflags option for go build here, interpolate the variable values
LDFLAGS=-ldflags "-X main.Version=${VERSION} -X main.Build=${BUILD}"

# Builds the project
build:
    go build ${LDFLAGS} -o ${BINARY}

# Installs our project: copies binaries
install:
    go install ${LDFLAGS}

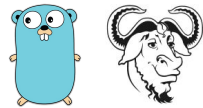
# Cleans our project: deletes binaries
clean:
    if [ -f ${BINARY} ] ; then rm ${BINARY} ; fi

.PHONY: clean install
```



Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ ls -al
total 4560
drwxr-xr-x  5 raul  staff    170 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff    306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff     19 25 Nov 03:11 Makefile
-rwxr-xr-x  1 raul  staff 2324096 25 Nov 05:08 gomake
-rw-r--r--  1 raul  staff    132 25 Nov 05:07 main.go
$ ./gomake
Version:  1.0.0
Build Time:  2015-11-25T05:23:32+0100
$ make clean
if [ -f gomake ] ; then rm gomake ; fi
$
```

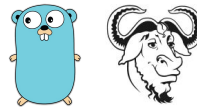


Make trivial to use compile time variables (III)

- Using LDFLAGS
 - The go link command allows you to set string variables at compile time with the -X option.
 - Use the format: `importpath.name=value`
 - <http://golang.org/cmd/link/>
- Differences between Golang 1.5+ and previous version.
 - Golang 1.5+ -X option takes one argument split on the first = sign.
 - Example:
 - `go build -ldflags "-X main.Version=1.0.0 -X main.Build=2015-11-25T00:23:32+0100"`
 - Golang < 1.5 -X options takes two separate arguments.
 - Example:
 - `go build -ldflags "-X main.Version 1.0.0 -X main.Build 2015-11-25T00:23:32+0100"`

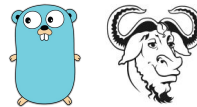
Reproducible builds

The image shows a perspective view down a long, brightly lit industrial corridor. On the left side, there are several tall metal shelving units filled with blue and white boxes, and a red and white fire extinguisher is visible. On the right side, there are long concrete counters with various items on top, including stacks of blue and white boxes. The floor is a smooth, light-colored concrete. In the background, there are large metal doors and some industrial equipment. The text "Reproducible builds" is overlaid in the center of the image in a large, white, sans-serif font.



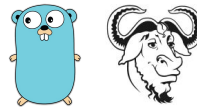
Reproducible builds.

- Delivering binaries is hard.
 - You just can't serve the binary over Internet.
 - Packages, installators there is plenty of tools to help you.
- You must be sure your binary is not modified or compromised.
 - Once the binary leaves your server you are not controlling it.
 - Also, a security attack can replace your binary on your server for a malicious one.
- shasum all your binaries!
 - Checking the finger-print of your binary allows you to ensure the binary has not been modified without permission.



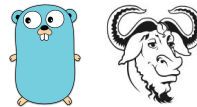
Reproducible builds (II).

- Our previous example introduces a new problem.
 - Using a timestamp variable changes the shashum of your file!
 - Just avoiding the build time fixes the problem.
 - But then, you're forced to bump the version on each build.
- Why not use the Git commit hash instead of the timestamp?
 - Can work together with the VERSION.
 - You are not anymore forced to bump a version on each build. Nightly builds!!
 - It doesn't change if the source code is not modified.
 - Identifies your binary, so you can know its "origin".



Demo

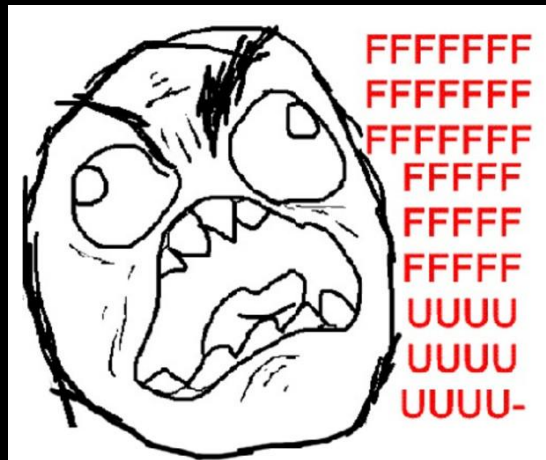
```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ shasum gomake
c6dd654ffe6f0e5af518d281da702187cc577cd4  gomake
$ make
go build
$ shasum gomake
dbf0cbe34067c42ecf6d221fcd789073370fa297  gomake
$
```



Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ shasum gomake
c6dd654ffe6f0e5af518d281da702187cc577cd4  gomake
$ make
go build
$ shasum gomake
dbf0cbe34067c42ecf6d221fcd789073370fa297  gomake
$
```

Different shasum values!





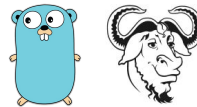
Final source code.

```
package main

import (
    "fmt"
)

// Variables to identify the build
var (
    Version string
    Build string
)

func main() {
    fmt.Println("Version: ", Version)
    fmt.Println("Git commit hash: ", Build)
}
```



Final Makefile

```
# This how we want to name the binary output
BINARY=gomake

# These are the values we want to pass for VERSION and BUILD
VERSION=1.0.0
BUILD=`git rev-parse HEAD`

# Setup the -ldflags option for go build here, interpolate the variable values
LDFLAGS=-ldflags "-X main.Version=${VERSION} -X main.Build=${BUILD}"

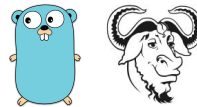
# Default target
.DEFAULT_GOAL: $(BINARY)

# Builds the project
$(BINARY):
    go build ${LDFLAGS} -o ${BINARY} ./..

# Installs our project: copies binaries
install:
    go install ${LDFLAGS} -o ${BINARY} ./..

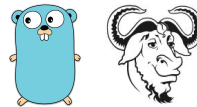
# Cleans our project: deletes binaries
clean:
    if [ -f ${BINARY} ] ; then rm ${BINARY} ; fi

.PHONY: clean install
```



Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ shasum gomake
b2c879490419f5a6473fa9d290f49a7e6a5b3353  gomake
$ make
go build
$ shasum gomake
b2c879490419f5a6473fa9d290f49a7e6a5b3353  gomake
$
```

Demo

```
$ ls -al
total 16
drwxr-xr-x  4 raul  staff  136 25 Nov 05:08 .
drwxr-xr-x  9 raul  staff  306 25 Nov 03:36 ..
-rw-r--r--  1 raul  staff   19 25 Nov 03:11 Makefile
-rw-r--r--  1 raul  staff  132 25 Nov 05:07 main.go
$ make
go build
$ shasum gomake
b2c879490419f5a6473fa9d290f49a7e6a5b3353  gomake
$ make
go build
$ shasum gomake
b2c879490419f5a6473fa9d290f49a7e6a5b3353  gomake
$
```

Same shasum values!
Reproducible builds!!



Thanks!

Slides & Code Examples : <https://github.com/repejota/gomake>

Questions?

