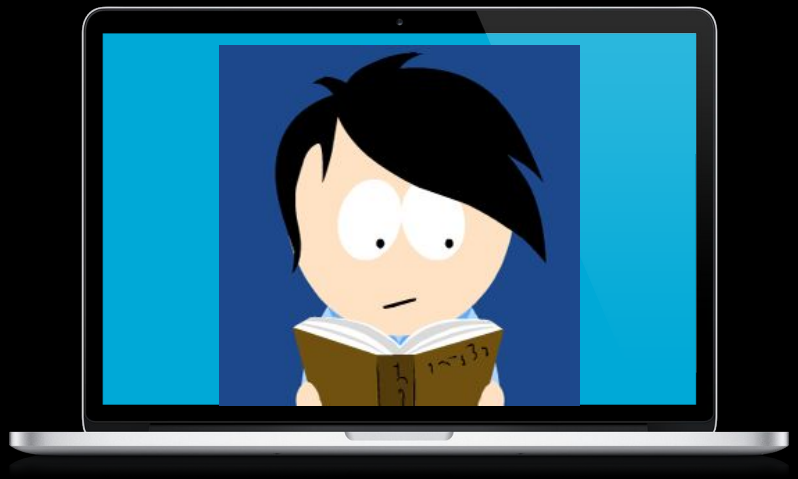
Refactoring to modules

Why, How and Everything Else

JFrog

Who am I?

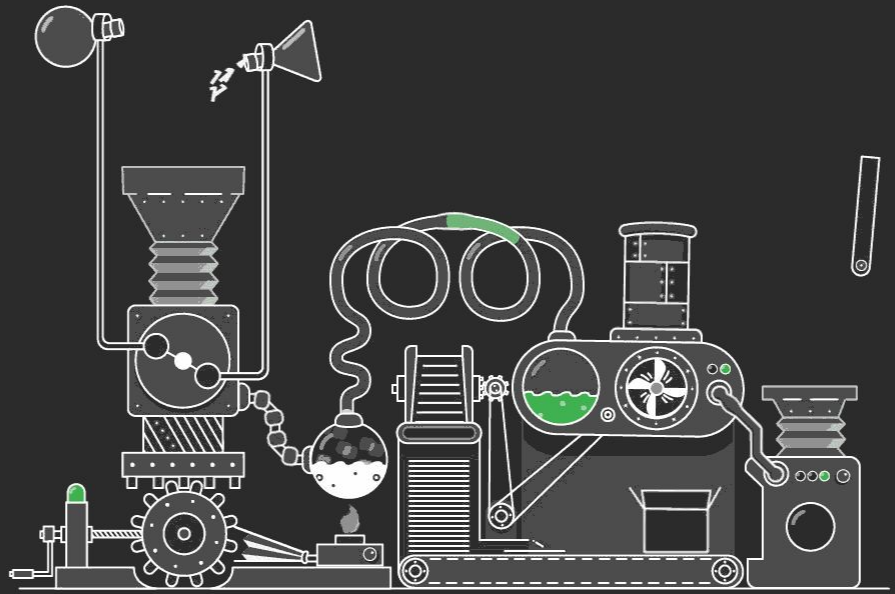


Elad Hirsch , DevOps Consultant

- DevOps Consultant
- Full-stack software engineer
- Tackling everything from **DevOps** and **Backend** challenges to the latest **Frontend** frameworks


JFrog
YOU LOOK FAMILIAR
memecrunch.com

JFrog Artifactory is getting ready to work...



JFrog

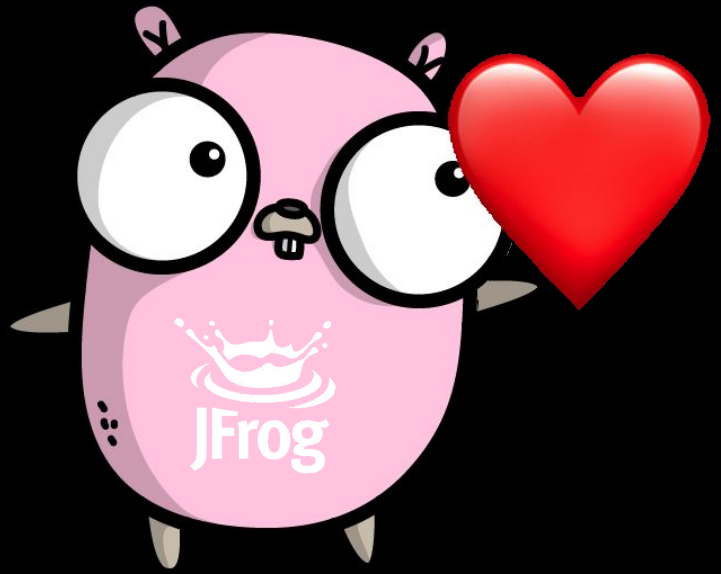
FROGS?


ANTS WORK··· FROGS PLAY
JFrog
*Artifactory Team
JOIN OUR PLAYGROUND


JFrog

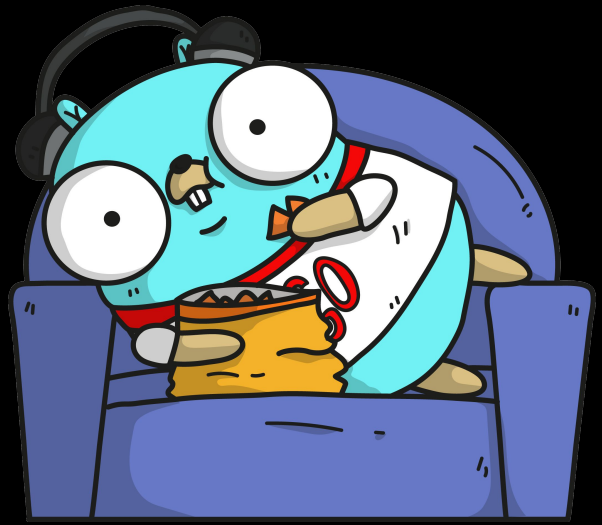
We absolutely love Go and we use it a lot

GoCenter
Xray
JFrog CLI
Metadata service
E+ Platform Installer



Let’s travel back in time…

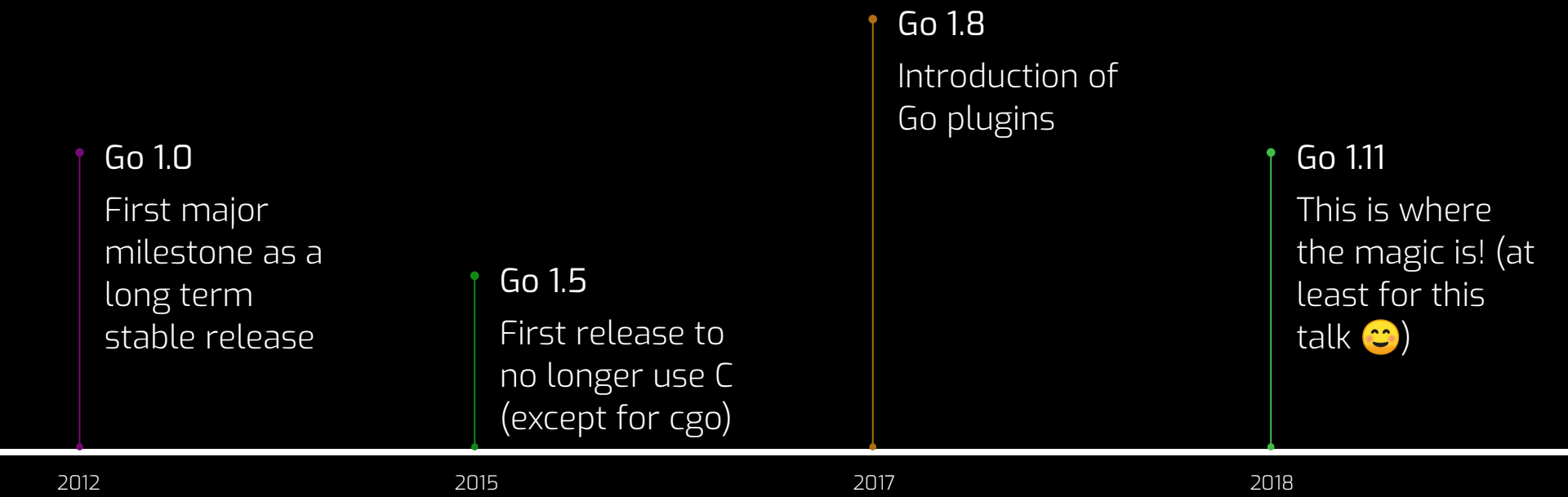
Let’s turn to the audience for a poll…



When did you start with Go?

1.0	1.2	1.5	1.8	1.11
2012	2013	2015	2017	2018

A quick history of go



So what is that magic?

Dependency Management...

"*Tis impossible to be sure of anything but Death and Taxes*"

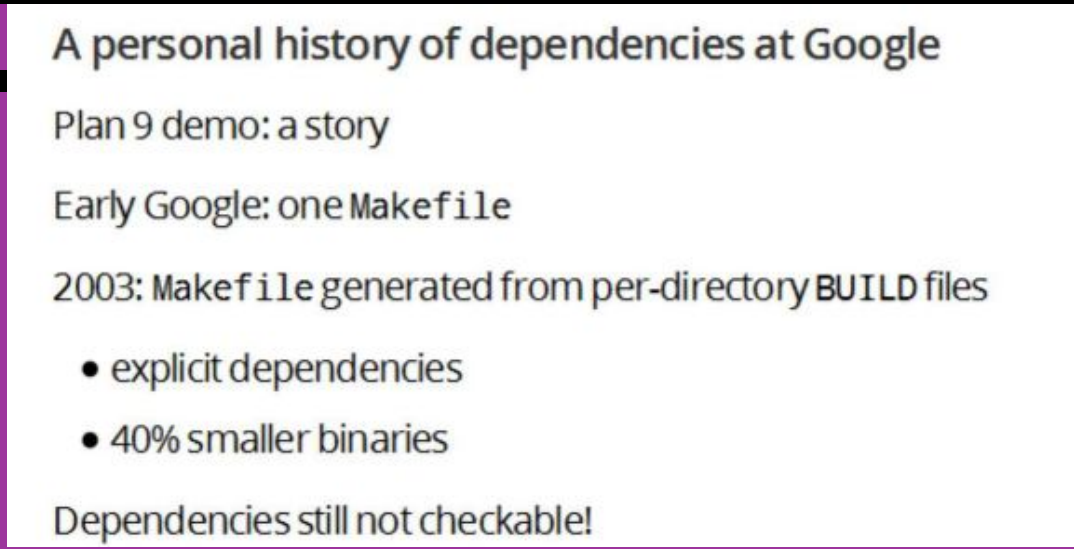
- Christopher Bullock

Let’s start with the why…

Why do we have this problem?

Source: Go at Google https://www.infoq.com/presentations/Go-Google

Why do we have this problem?



Source: Go at Google https://www.infoq.com/presentations/Go-Google

Let’s go fix it!

One easy solution…?

Dependencies are sources!

Remote imports are in VCS

Dump everything into a single folder

Compile…

Profit!!

Wait ! But… how do I know…

- Which dependencies I use?
- Which dependencies you used?
- Which dependencies I should use?
- Which code I'm editing right now?
- Is there any dependency duplications?
- What is going on?!

Let’s duplicate dependencies…

“Check your dependencies to your own VCS.”

Vendoring – the worst kind of forking

"Copy all of the files at some version from one version control repository and paste them into a different version control repository"

But what is wrong with vendoring?

- History, branch, and tag information is lost
- Pulling updates is impossible
- It invites modification, divergence, and bad fork
- It wastes space
- Good luck finding which version of the code you forked

Build your own dependency manager…

"It's not the role of the tooling provided by the language to dictate how you manage your code (...)"

Andrew Gerrand


AND THE NEXT THING YOU KNOW...
LIGHTYEAR
THERE ARE 19 DEPENDENCY MANAGERS

So you want to write a package manager

You woke up this morning, rolled out of bed, and thought, "Y'know what? I don't have enough misery and suffering in my life. I know what to do—I'll write a language package manager!"

...

Package management is awful, you should quit right now

Package management is a nasty domain. Really nasty. On the surface, it *seems* like a purely technical problem, amenable to purely technical solutions. And so, quite reasonably, people approach it that way. Over time, these folks move inexorably towards the conclusion that:

1. software is terrible
2. people are terrible
3. there are too many different scenarios
4. nothing will really work for sure
5. it's provable that nothing will really work for sure
6. our lives are meaningless perturbations in a swirling vortex of chaos and entropy

★ Top highlight

Go dep – Proper dependency management?

- Working in project directories
- Local cache for dependencies
- Version declarations
- Conflict resolution

And according to the dep project page:

> *dep was the "official experiment." The Go toolchain, as of 1.11, has (experimentally) adopted an approach that sharply diverges from dep. As a result, we are continuing development of dep, but gearing work primarily towards the development of an alternative prototype for versioning behavior in the toolchain.*

For more information about the new Go build-in management, please refer to the official GitHub Wiki — Go 1.11 Modules.

Thanks John Arundel @bitfield and Erhan Yakut @yakuter for revealing the problem. 🙇

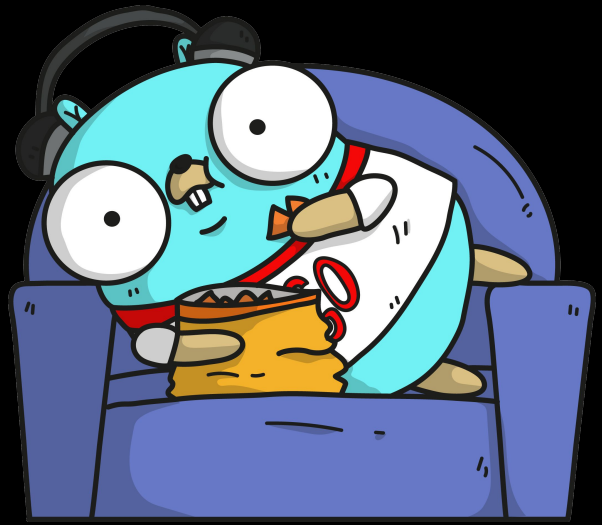
—

Update @ 2018–02–03: Sam Boyer from the godep team has clarified some incorrect information in this article. I apologize to Sam Boyer and the readers for any inconvenience. 😢


icanhasGIF.com
Now what?

Let’s actually go fix it!

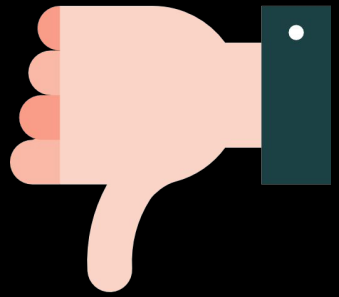
Let’s turn to the audience for another poll…



Who is using Go modules?

Yes

No



Let’s go over some definitions

Module

A module is a collection of related Go packages that are versioned together as a single unit.

Let’s go over some definitions

Sources

A module is a tree (directory) of Go source files with a `go.mod` file in the root directory.

Let’s go over some definitions

Version Control

Most often, a single version-control repository corresponds exactly to a single module

Using go modules

Set the environment variable
`GO111MODULE` to `ON`

Use go outside of your
`$GOPATH`

Creating a module

```
mkdir mymodule
cd mymodule
go mod init github.com/retgits/mymodule
```

Moving from DEP to modules

- Import dependencies from `Gopkg.lock`

```
go mod init <module path>
```

- Remove unnecessary imports and add indirect imports

```
go mod tidy
```

- Delete the vendor folder

```
rm –rf vendor/
```

- Delete Gopkg files

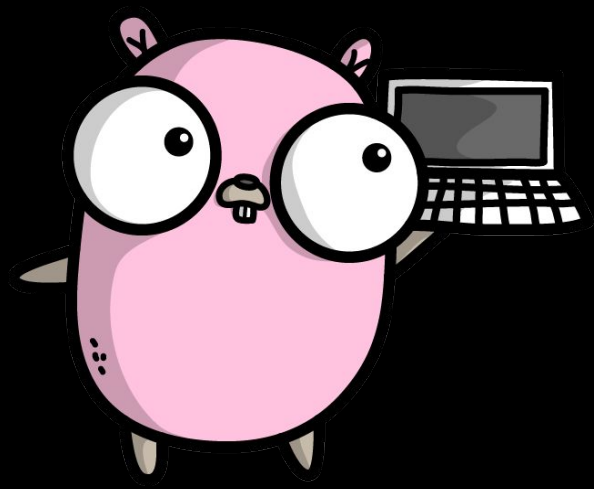
```
rm Gopkg.*
```

Demo time …

Go.mod by any other name…
Would probably not work…

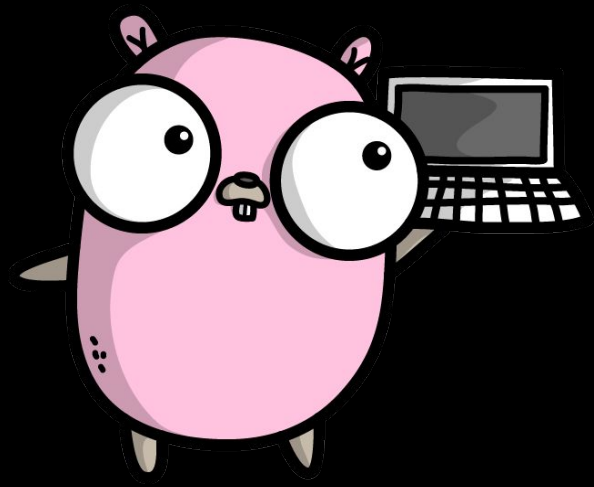
versioning

```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)
```



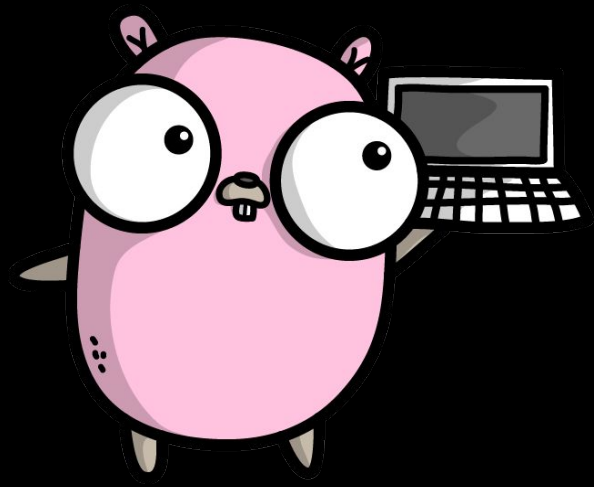
Replacing packages

```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)

replace github.com/some/dependency github.com/retgits/dependency
```



Works with local folders too!

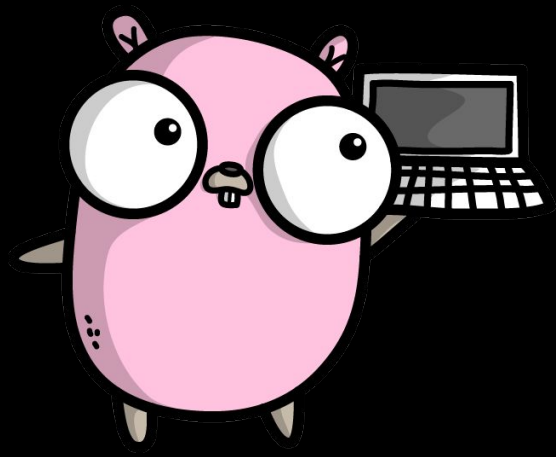
```
module github.com/retgits/mymodule

go 1.12

require (
        github.com/naoina/go-stringutil v0.1.0
        github.com/some/dependency v1.2.3
        github.com/google/go-github/v25 v25.0.1
)

replace github.com/some/dependency ../../Downloads/dependency
```



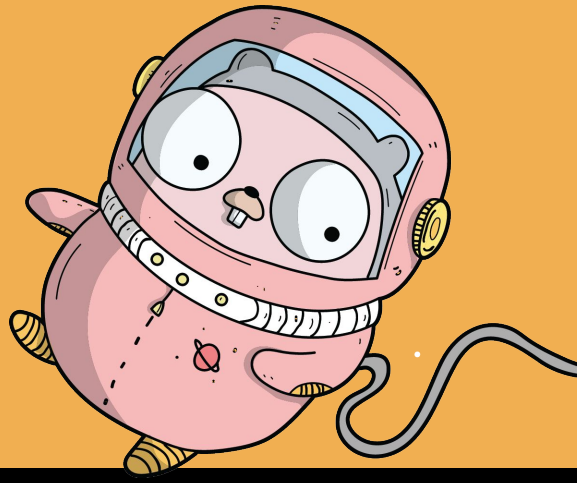
But I kinda like the vendor folder?

- You can create the vendor folder
  ```
  go mod vendor
  ```
- You can use the vendor folder during builds
  ```
  go build -mod=vendor
  ```

This does mean you need to have all dependencies listed in your go.mod file



Houston… I think I lost my module?



Modules are immutable

The **<module>@v<version>** construct should be immutable

That means that

`github.com/retgits/checkiday/@v/v1.0.0`

Should forever be the same...

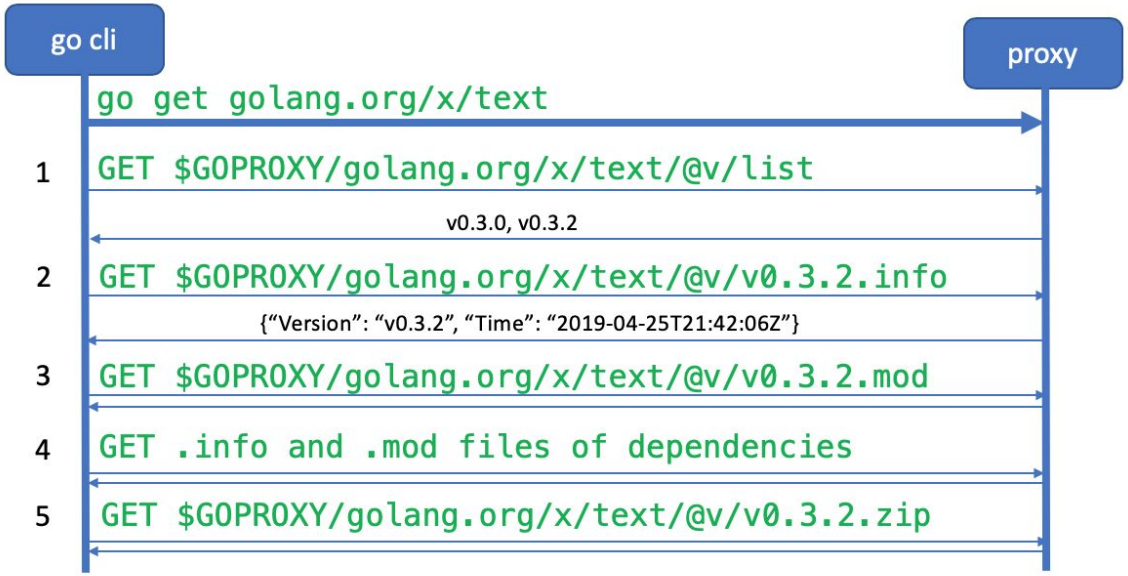
But are they really?

"Friends don't let friends do git push -f"

- Aaron Schlesinger

Using the *GOPROXY* variable

```
export GOPROXY=https://myawesomeproxy.com
go get github.com/retgits/checkiday
```



$GOPATH/pkg/mod/cache/download/golang.org/x/text/@v/

list	v0.3.0.info	v0.3.0.zip	v0.3.2.info	v0.3.2.mod	v0.3.2.ziphash
list.lock	v0.3.0.mod	v0.3.0.ziphash	v0.3.2.lock	v0.3.2.zip	

$GOPATH/pkg/mod/golang.org/x/text@v0.3.2/

Keeping modules

Local cache (**`$GOPATH/pkg/mod`**)

Immediate access, not shared, can be wiped...

Organizational cache (private proxy)

Fast access, requires infra, shared across devs

Public cache (public proxy)

Highly available, CDN, no infra, free

THE Athens PROJECT

JFrog GoCenter

And also faster builds…

JFrog GoCenter The Central Go Modules Repository

ReadMe

Add

106,797

Versions indexed

Feedback

Contact Us Privacy Policy Terms Of Service Status

Go get my a new super duper module

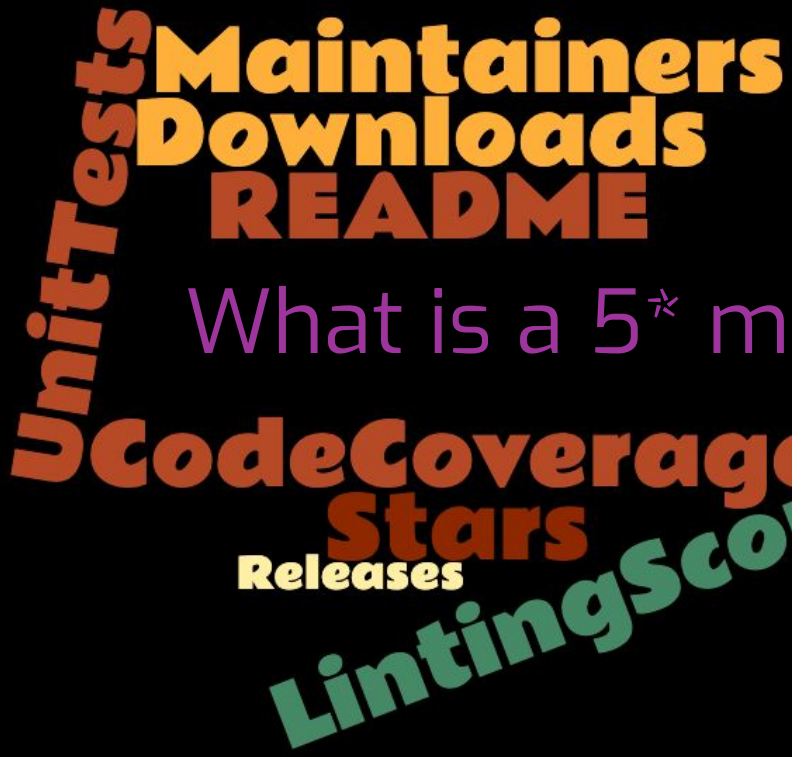
Before modules

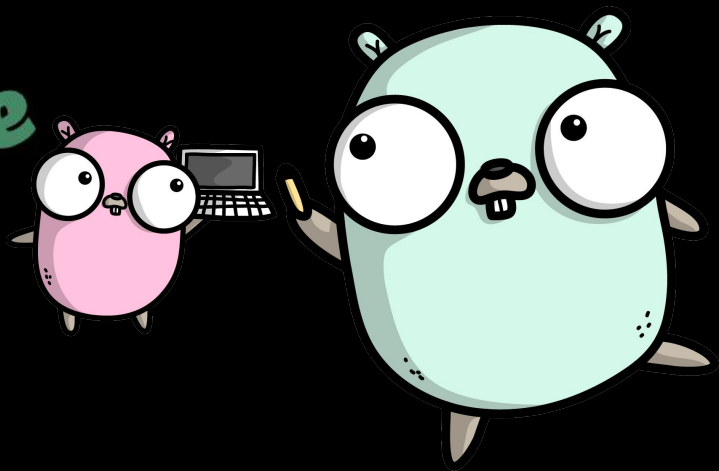
```
go get https://github.com/sirupsen/logrus
git clone https://github.com/sirupsen/logrus
<build module locally>
```

With modules

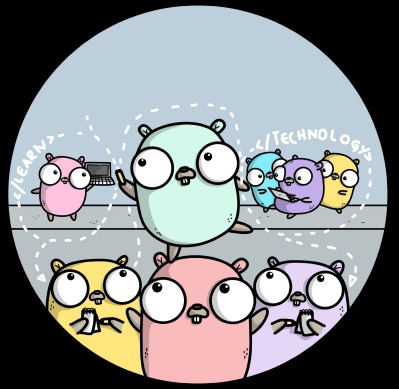
```
go get github.com/sirupsen/logrus
GET github.com/sirupsen/logrus/@v/list
GET github.com/sirupsen/logrus/@v/v1.0.0.mod
GET github.com/sirupsen/logrus/@v/v1.0.0.zip
```

How can we ensure module quality ?





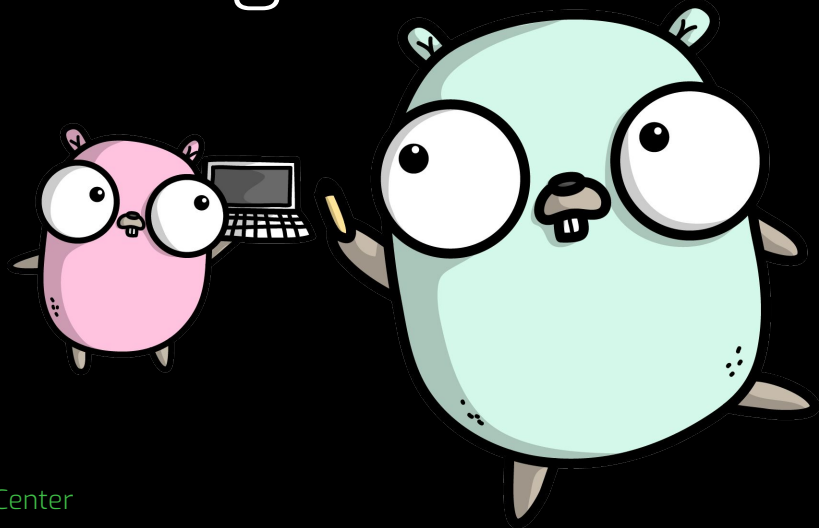
Final thoughts



- If you haven't, start with Go modules:
  `GO111MODULE=on`
- Start working outside of your **`$GOPATH`**
- Use a public proxy server for immutable Go modules (l**ike GoCenter.io**)
- Think about running your own proxy server or repository manager such as **Artifactory** or **Athens**

But before I go…

There's one more thing…

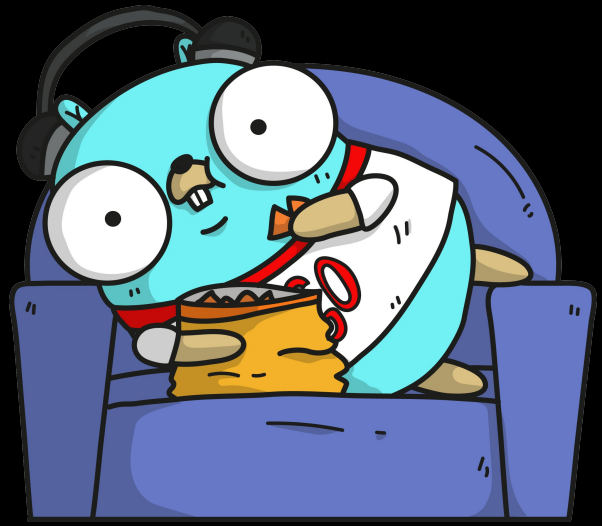


JFrog CLI - GO support

```
// Configure Artifactory:
jfrog rt c

//Configure the project's repositories::
jfrog rt go-config

//Build the project with go and resolve the project dependencies from
Artifactory.
jfrog rt go build --build-name=my-build --build-number=1

//Publish the package we build to Artifactory.
jfrog rt gp go v1.0.0 --build-name=my-build --build-number=1

//Collect environment variables and add them to the build info.
jfrog rt bce my-build 1

//Publish the build info to Artifactory.
jfrog rt bp my-build 1
```

Twitter, ads, and Q&a

- https://jfrog.com/shownotes
- @JFrog
- #GoLang
- https://gocenter.io



Thank you!