Myntra

APIfy Talks - Talk 1

What's happening with Go Modules

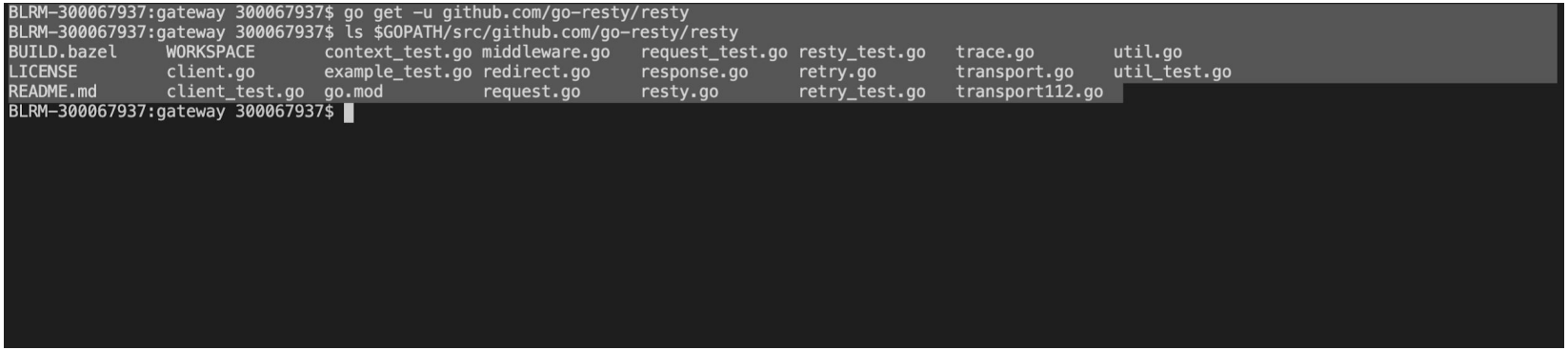
By Salman Shah

Myntra

History

The idea behind go get

- No versioning or dependency



History

Gopkg - Workaround for the problem

- https://labix.org/gopkg.in

Myntra

History

Glide - Open source solution

- Golang 1.5+ shipped with the vendor experiment

Issues:

- One of many package managers
- Certain issues at the beginnning with hard dependency management rules

Myntra

Dep, a stable solution

Engagement with the GoTeam to support it officially by Sam Boyer

➔ **Similarity**

Install/update similar to Node/Ruby dependency management

➔ **Official Support**

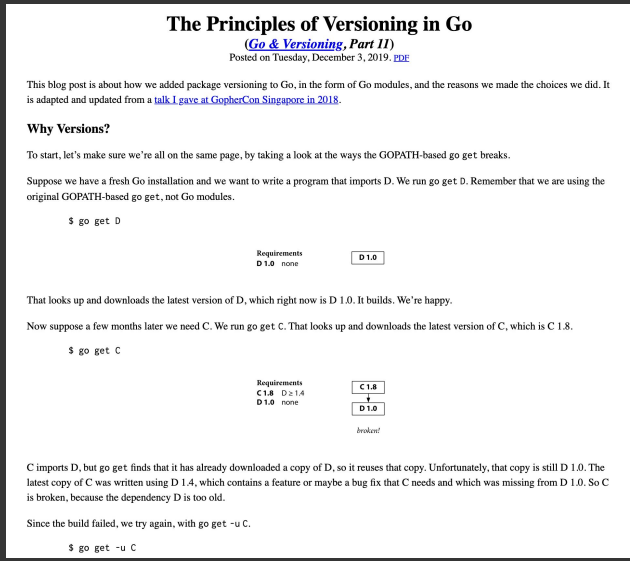
Unlike its predecessors, dep was officially supported.

➔ **Vendoring**

Dependency resolution was more accurate

Enter vgo! Also known as go mod in later releases

(With faster and accurate dependency resolutions)



The Principles of Versioning in Go

(*Go & Versioning, Part 11*)

Posted on Tuesday, December 3, 2019. PDF

This blog post is about how we added package versioning to Go, in the form of Go modules, and the reasons we made the choices we did. It is adapted and updated from a talk I gave at GopherCon Singapore in 2018.

Why Versions?

To start, let's make sure we're all on the same page, by taking a look at the ways the GOPATH-based go get breaks.

Suppose we have a fresh Go installation and we want to write a program that imports D. We run go get D. Remember that we are using the original GOPATH-based go get, not Go modules.

```
$ go get D
```

Requirements
D 1.0 none

D 1.0

That looks up and downloads the latest version of D, which right now is D 1.0. It builds. We're happy.

Now suppose a few months later we need C. We run go get C. That looks up and downloads the latest version of C, which is C 1.8.

```
$ go get C
```

Requirements
C 1.8 D ≥ 1.4
D 1.0 none

C 1.8
↓
D 1.0

broken!

C imports D, but go get finds that it has already downloaded a copy of D, so it reuses that copy. Unfortunately, that copy is still D 1.0. The latest copy of C was written using D 1.4, which contains a feature or maybe a bug fix that C needs and which was missing from D 1.0. So C is broken, because the dependency D is too old.

Since the build failed, we try again, with go get -u C.

```
$ go get -u C
```

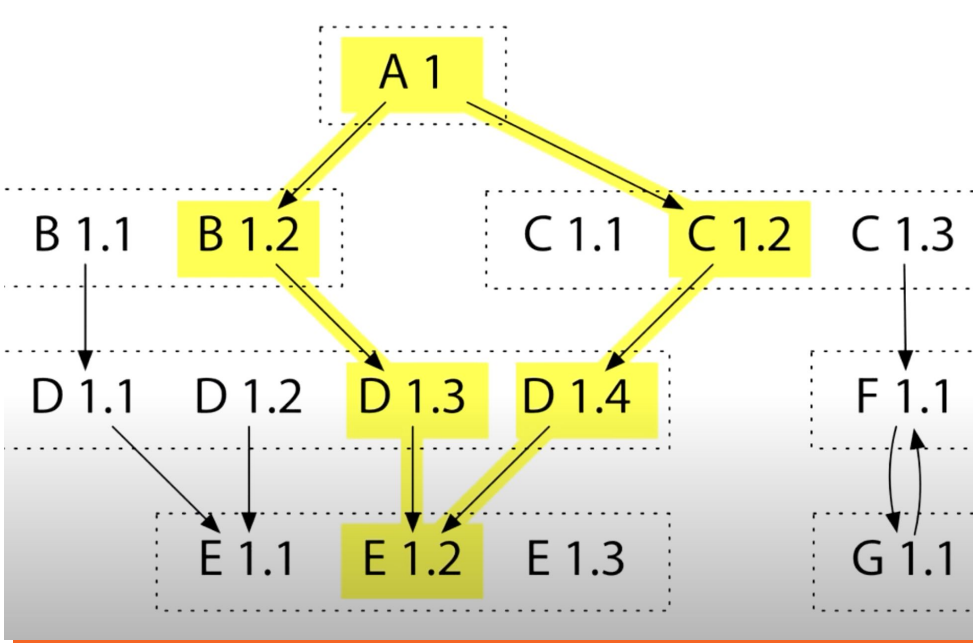
Dependency Management

About

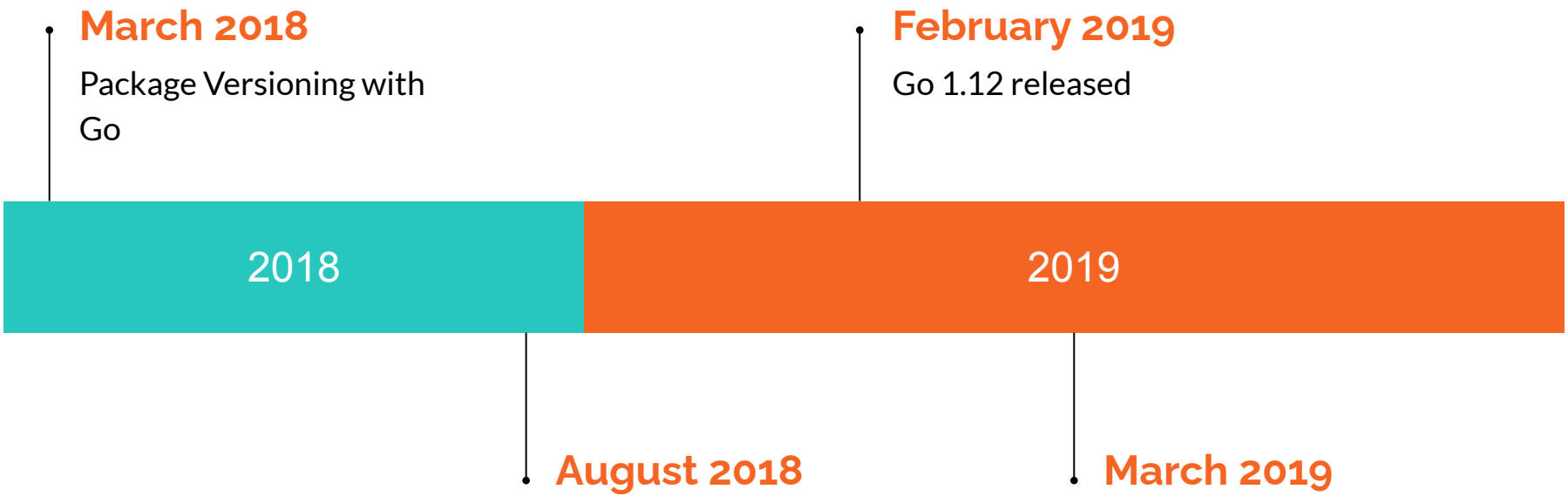
Dependency Management np-complete traditionally

Ex: npm install

Vgo tried to beat that minimum version selection algorithm.



Milestones


March 2018
Package Versioning with Go
February 2019
Go 1.12 released
2018
2019
August 2018
Go 1.11 released
Experimental modules release
March 2019
Go modules officially released by Go Team

Examples

Golimit