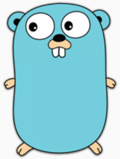
Programming in Go

Matt Holiday
Christmas 2020



Go Modules

Why modules?

Go module support is intended to solve several problems:

- avoid the need for `$GOPATH`
- group packages versioned/released together
- support semantic versioning & backwards compatibility
- provide in-project dependency management
- **offer strong dependency security & availability**
- continue support of vendoring
- **work transparently across the Go ecosystem**

Go modules with proxying offers the value of vendoring without requiring your project to vendor all the 3rd-party code in your repo

Why modules?

Go's dependency management protects against some risks:

- flaky repos
- packages that disappear
- conflicting dependency versions
- surreptitious changes to public packages

But it cannot ensure the actual quality or security of the *original* code; see

- Reflections on Trusting Trust by Ken Thompson
- Our Software Dependency Problem by Russ Cox

"A little copying is better than a little dependency" — Go Proverb

Import compatibility rule

“If an old package and a new package have the same import path,
the new package must be backwards compatible with the old package”

An *incompatible* updated package should use a new URL (version)

```
package hello

import (
       "github.com/x"
    x2 "github.com/x/v2"
)
```

Note that you can import both versions if necessary

Some control files

The `go.mod` file has your module name along with direct dependency requirements (and from Go 1.13, the version of Go)

```
module hello
require github.com/x v1.1
go 1.13
```

The `go.sum` file has checksums for all *transitive* dependencies

```
github.com/x v1.1 h1:KqKTd5BnrG8aKH3J...
github.com/y v0.2 h1:Qz0iS0pjZuFQy/z7...
github.com/z v1.5 h1:r8zfno3MHue2Ht5s...
```

Always check them in to your repo

Some environment variables

We typically use the defaults for these

```
GOPROXY=https://proxy.golang.org,direct
GOSUMDB=sum.golang.org
```

and set this for private repos

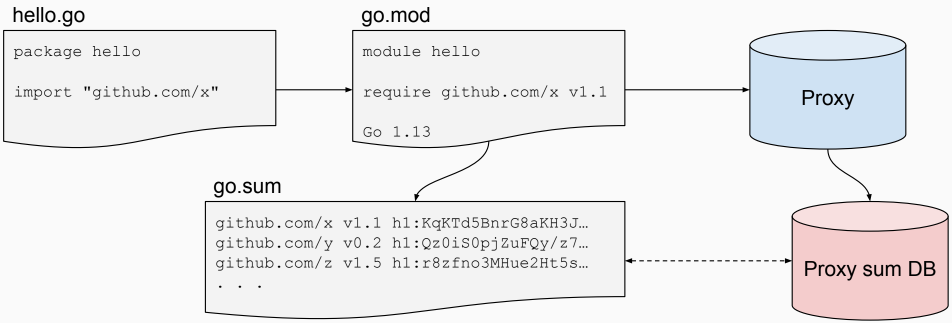
```
GOPRIVATE=github.com/xxx,github.com/yyy
GONOSUMDB=github.com/xxx,github.com/yyy
```

Remember also you must be set up for access to private Github repos in order to download private modules

Module Proxy

The `go.mod` file records direct dependencies, while `go.sum` records checksums for all (transitive) dependencies

The proxy caches modules and keeps a secure history tree



```
package hello

import "github.com/x"
```

```
module hello

require github.com/x v1.1

Go 1.13
```

```
github.com/x v1.1 h1:KqKTd5BnrG8aKH3J…
github.com/y v0.2 h1:Qz0iS0pjZuFQy/z7…
github.com/z v1.5 h1:r8zfno3MHue2Ht5s…
. . .
```

Some details

The `go.mod` file may record pseudo-versions (for non-release/trunk versions) as well as “replacements”

```
require (
    cloud.google.com/go v0.35.1
    github.com/gen2brain/malgo v0.0.0-20181117112449-af6b9a0d538d
    github.com/gorilla/context v1.1.1 // indirect
    github.com/gorilla/mux v1.6.2
    github.com/gorilla/websocket v1.4.0
    github.com/satori/go.uuid v1.2.0
    golang.org/x/net v0.0.0-20190119204137-ed066c81e75e
    google.golang.org/api v0.1.0
    google.golang.org/genproto v0.0.0-20190123001331-8819c946db44
    google.golang.org/grpc v1.18.0
    gopkg.in/yaml.v2 v2.2.2
)

replace github.com/satori/go.uuid v1.2.0 =>
      github.com/satori/go.uuid v1.2.1-0.20181028125025-b2ce2384e17b
```

Maintaining dependencies

Start a project with

```
$ go mod init <module-name>   ## create the go.mod file
$ go build                    ## building updates go.mod
```

Once a version is set, Go will not update it automatically; you can update every dependency with

```
$ go get -u ./...             ## update transitively
$ go mod tidy                 ## remove unneeded modules
```

You **must commit** the `go.mod` and `go.sum` files in your repo

Maintaining dependencies

You can list available versions of a dependency

```
$ go list -m -versions rsc.io/sampler
```

There are several ways to update a single dependency

```
$ go get github.com/gorilla/mux@latest
$ go get github.com/gorilla/mux@v1.6.2
$ go get github.com/gorilla/mux@e3702bed2
$ go get github.com/gorilla/mux@c856192     ## non-release
$ go get github.com/gorilla/mux@master      ## non-release
```

You **must commit** the `go.mod` and `go.sum` files in your repo

Vendoring and the local cache

Use `go mod vendor` to create the vendor directory; it must be in the module's root directory (along with `go.mod`)

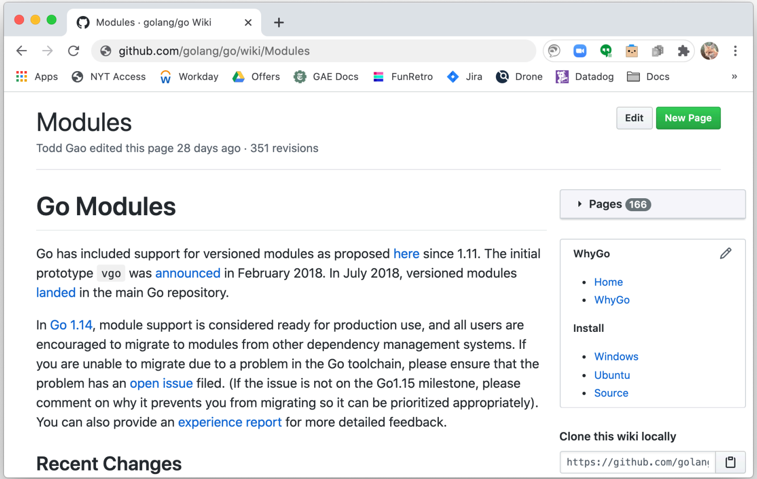
In Go 1.13, you must use `go build -mod=vendor` to use it (not required 1.14+)

Go keeps a local cache in `$GOPATH/pkg`

- each package (using a directory tree)
- the hash of the root checksum DB tree

Use `go clean -modcache` to remove it all (i.e., in make clean)

More info



Modules

Todd Gao edited this page 28 days ago · 351 revisions

Go Modules

Go has included support for versioned modules as proposed here since 1.11. The initial prototype `vgo` was announced in February 2018. In July 2018, versioned modules landed in the main Go repository.

In Go 1.14, module support is considered ready for production use, and all users are encouraged to migrate to modules from other dependency management systems. If you are unable to migrate due to a problem in the Go toolchain, please ensure that the problem has an open issue filed. (If the issue is not on the Go1.15 milestone, please comment on why it prevents you from migrating so it can be prioritized appropriately). You can also provide an experience report for more detailed feedback.

Recent Changes