Getting Started with Go Modules

Women Who Go NYC

Agenda

- Go and Dependency Management
- Introduction to Go Modules and SemVer
- Common commands and tasks
- Hands on walk through setting up Go Modules
- Migrating to Go Modules
- Athens: A Go module datastore and proxy

Go Dependency Management

Go was released without a dependency management solution.

The community development several approaches over the years:

- GoDep
- GoVendor
- Glide
- Vendoring
- Dep

Go Modules, released as opt-in in 1.11, is the recommended solution for dependency management in Go, to be fully released in 1.13.

Why Go Modules?

- Encourage developers to tag releases of their packages. Tagging explicit releases makes clear what is expected to be useful to others and what is still under development.
- Move away from 3rd party version control tools which are unavailable to users who cannot or choose not to install these tools.
- Enable multiple modules to be developed in a single source code repository but versioned independently.
- Make it easy to put caching proxies in front of dependency downloads, whether for availability or security reasons.
- Make it possible to introduce a shared proxy for the Go community.

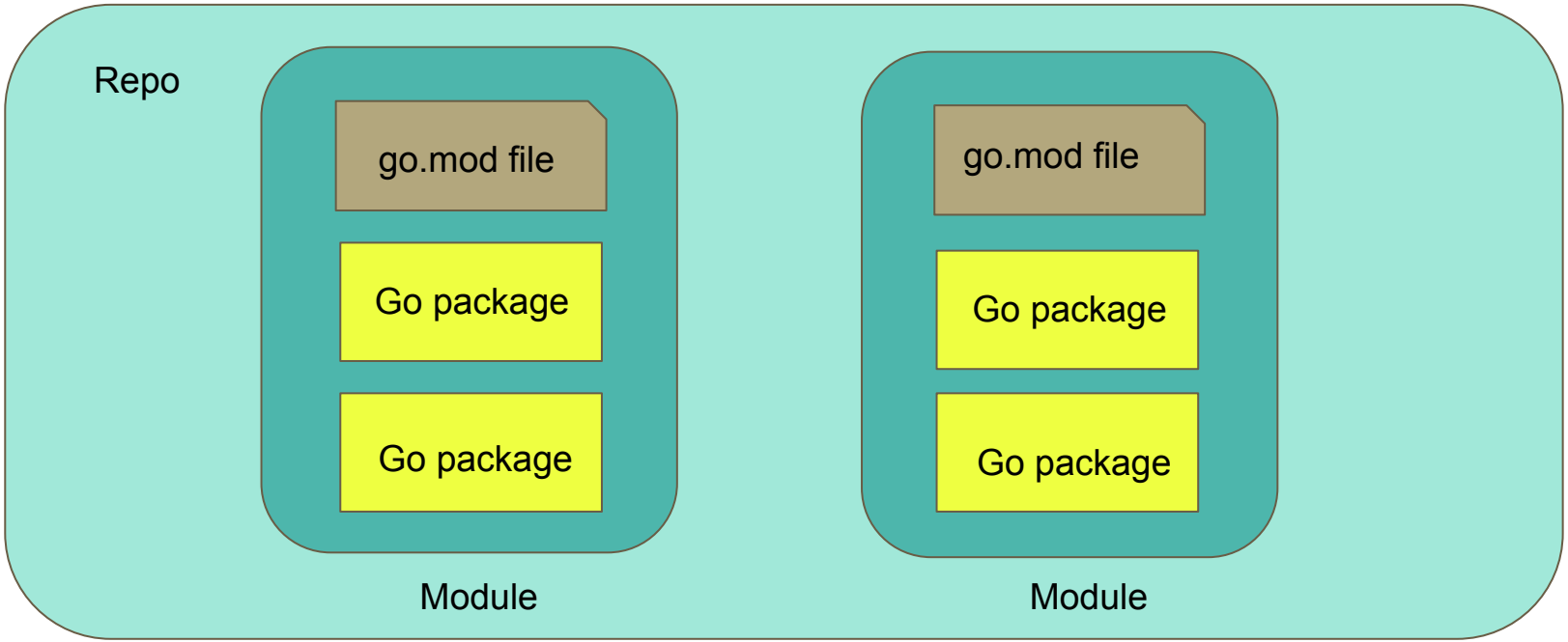
What is a Go Module?

A Go module is a collection of packages versioned as a unit, and contains a go.mod file listing all other required modules.

Modules record precise dependency requirements and create reproducible builds. Summarizing the relationship between repositories, modules, and packages:

- A repository contains one or more Go modules.
- Each module contains one or more Go packages.
- Each package consists of one or more Go source files in a single directory

What is a Go Module?



Introducing SemVer

What?

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

Why?

A simple set of rules for how versions are assigned and incremented.

Versions convey meaning about the underlying code and what has changed from on version to the next.

Developers can specify and control the dependencies of the software.

Introducing Semver

Go Modules must be semantically versioned according to semver in the form.

- If the module is **version v2 or higher**, the major version of the module must be included as a /vN at the end of the module paths
- If the module is **version v0 or v1**, do not include the major version in either the module path or the import path.
- If you add a new import not yet covered by a require in go.mod, the **highest version** of that dependency will be selected.
- The minimal version selection algorithm is used to select the versions of all modules used in a build. For each module in, the version selected is the semantically highest of the versions explicitly listed by a require directive in the main module or one of its dependencies.

Go Modules Key Takeaways

- Vendor folder is ignored. The go command downloads from the dependency sources and uses those downloaded copies (defaulting to $HOME/go/src/mod).
- Go uses SemVer to select the latest version when adding new modules.
- When fetching module's dependencies, Go modules will use the specific dependency versions requested by that module.
- If a repository does not have any valid semver tags, then the repository's version will be recorded with a "pseudo-version" (such as v0.0.0-20171006230638-a6e239ea1c69).

Using Go Modules

1. Navigate to the root of the module's source tree outside of GOPATH.
2. Create the initial module definition and write it to the go.mod file.

```
$ go mod init
```

3. Build the module, which will automatically add missing dependencies.

```
$ go build ./...
```

4. Test the module as configured to ensure that it works with the selected versions.

```
$ go test ./...
```

Hands On

Step by step instructions at: https://github.com/Yigenana/WWGNYC

If you do not have Go 1.11 installed locally, install Docker

Mac: https://docs.docker.com/docker-for-mac/install/

Windows: https://docs.docker.com/docker-for-windows/install/

go.sum file

For validation purposes, go.sum contains the expected cryptographic checksums of the content of specific module versions.

go.sum provides additional security that module content for a specific tag has not changed.

Contains checksums for modules you've stopped using. This allows validation of the checksums if you later resume using something, which provides additional safety.

It's recommended to commit your go.sum file.

Migrating to Go Modules

- ‘go mod init’ automatically translates the required information from dep, glide, govendor, godep and 5 other pre-existing dependency managers.
- If you are creating a v2+ module, be sure your module directive in the converted go.mod includes the appropriate /vN.
- If you are importing v2+ modules, you might need to do some manual adjustments in order to add /vN to the require statements that go mod init generates after translating from a prior dependency manager.
- go mod init will not edit your .go code to add any required /vN to import statements.
- Older versions of Go understand how to consume a vendor directory created by ‘go mod vendor’ command.