How to become a Go open source developer

SKG Gophers - Panagiotis Georgiadis

whoami

Panagiotis Georgiadis
drpaneas

Edit profile

46 followers · **2** following · ☆ **78**

Red Hat
Nuremberg, Germany

Overview | Repositories 166 | Projects | Packages

drpaneas / README.md — Send feedback

Hello, my friend 👋 stay awhile and listen...

Greetings, my name is **Panagiotis Georgiadis**, people call me **Panos** or `drpaneas`. I'm an open-source developer from Greece, living in Germany. I'm currently working as Software Engineer & SRE for OpenShift at RedHat and before that I used to work as QA & Software Engineer at SUSE. Outside of work, I'm interested in music, astrophysics, movies, comics, cooking, video games and blockchain.

⚡ Focus

Linux | Go | Kubernetes | Docker | Amazon | Google Cloud Platform

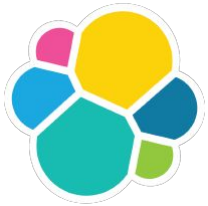
📫 How to reach me

LinkedIn | PeerTube | @drpaneas:boobalar.net | Follow @drpaneas 7 | Follow @panosgeorgiadis 264

My code contributions



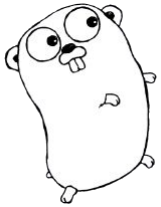




















Pre-Req

It goes without saying, but ...

- A computer (preferrably Linux or Mac)
- Good enough English skills
- Internet connection
- Basic Go skills
- Basic git skills

- Ability to learn on your own
- Self-motivated
- Continuous learner
- Communication skills
- Sharing knowledge
- Team player
- Curiosity

Dev Env Setup

Go installation

- It’s just a binary ...
- Do NOT use package managers

Configure:

- `mkdir -p $GOPATH/{bin,pkg,src}`

- GOPATH
- GOBIN
- PATH

Every time your create a project:

- `cd $GOPATH/src`
- `mkdir foo; cd foo`
- `git init -q`
- `go mod init`

IDEs

- Code style: Code Highlighting, auto-indentation
- Preview signature and docs
- Go to definition/implementation
- Auto-completion or suggestions
- Auto-import libraries
- Refactoring mechanisms
- Debugging
- Testing
- Code Generators
- git features (e.g. git blame)
- Pair-Programming
- Remote

Goland

VSCode + extensions

emacs + plugins

vim + plugins

Makefiles

- **Build:** `go build`
- **Setup:** `go mod download -x; go generate -v ./...`
- **fmt:** `go imports -w ${SRC}, go gmt -w -s ${SRC}`
- **test:** `go test -failfast -race -coverpkg=./... -covermode=atomic -coverprofile=coverage.txt ./... -run . -timeout $(TIMEOUT)m`
- **cover:** `go tool cover -html=coverage.txt`
- **lint:** `golangci-lint run --max-issues-per-linter 0 --max-same-issues 0 ./...`
- **spell-check:** `misspell ./...`
- **clean:** `go clean; rm $files (or .gitignore)`
- **todo:** `@grep -I -R -n TODO * --exclude=Makefile`
- **run:** `go run $TARGET`
- **serve:** `@docker run --rm -it -p 8000:8000 -v ${PWD}:/docs squidfunk/mkdocs-material`
- **help:** `@sed -n 's/^##//p' ${MAKEFILE_LIST} | column -t -s ':' | sed -e 's/^/ /'`
- and others ... (e.g. `make generate`)

Makefiles or GH Actions?

- Do NOT rely on local versions (set a hardcoded value)
- Use containers for everything (only dependency would be docker or podman)

GH Actions (or any other CI):

- Use your makefile or ... do not use a makefile at all

GH Actions only:

- Find the equiv. GH Action for every target
- Use **act** locally

Automation
Devops
CI/CD
Release

CI/CD

- Set it up before start coding
- You need to be admin on the repo.
- Ask your organization if they already do some of that stuff.

Once you’re done, you can fork and use it for every single of your Go repositories.

CI/CD - Create a Makefile

Description

Every project needs ones. Read https://opensource.com/article/18/8/what-how-makefile

Acceptance Criteria

1. Create a Makefile for Go projects (see similar ones to get inspired)
2. setup: section runs go mod download and go generate -v ./...
3. test: section runs all the tests with coverage (saved into coverage.txt) and timeouts in 5m`
4. cover: runs all the tests and opens the coverage report
5. fmt: runs gofmt and goimports in all go files
6. lint: run all the linters (that is golangci-ling and mispell)
7. ci: run all the tests and code checks
8. build: go build it
9. todo: show all the todo strings per file
10. the default choice should be build target

CI/CD - Configure golangci-lint and misspel

Description

Both golangci-lint and misspel are known to run in various CI systems against Golang projects. Make sure they are available to run (e.g. binaries are available to get executed).

Acceptance Criteria

1. Create the appropriate section in the Makefile called lint running `golangci-lint run ./...` and `misspell -error **/*`
2. Make sure the Makefile runs go mod download and `go generate -v ./...`
3. Add the appropriate `//go:generate` commands into main.go
4. Use the tools.go paradigm
5. Running the makefile should run the linters and generate the new entries in `go.sum` and `go.mod` respectively.

CI/CD - Multi-platform testing

Description

The sooner we implement integration testing in Linux, Windows and MacOS, the sooner we can start coding. Create a CI pipeline that runs on push and PRs and builds the software (later we will add tests on that).

Acceptance Criteria

1. Implement a multiplatform GH Action workflow. Maybe this: https://github.com/marketplace/actions/go-cross-build can help
2. Should run on latest ubuntu 20.04 (see https://docs.github.com/en/actions/reference/specifications-for-github-hosted-runners#supported-runners-and-hardware-resources)
3. Make sure the artifacts are uploaded
4. As as dummy test, just run the binary and check for the exit code.

CI/CD - Automated builds and releases

Description

Whenever there is a new release, we want to automatically build and release. Create the bare minimum configuration and pipeline to achieve that for a Go project that is using go modules.

Acceptance Criteria

1. Use https://github.com/marketplace/actions/goreleaser-action
2. Create the `.goreleaser.yml`
3. Create a build workflow that builds and releases the artifacts. It gets triggered when:

```
on:
  push:
    branches:
      - 'master'
    tags:
      - 'v*'
  pull_request:
```

CI/CD - Create changelog

Description

It's a common practice to have a changelog, even though nobody reads it. But well ...

Using goreleaser:

1. Sort all the commits since the last release
2. Exclude commits that start in the title with docs or test or Merge pull request or Merge branch or go mod tidy.

** or alternatively** try this:

Acceptance Criteria

1. Install https://github.com/marketplace/actions/changelog-ci
2. Test it

CI/CD - Bot for auto merging

Description

We need a bot that will either block or allow merging based on some conditions. In some rare cases, the bot will automatically merge PRs created by bots.

Acceptance Criteria

1. Install and read about kodiak bot
2. Look for similar configuration in Github and post them here
3. Configure it to merge only if there are 2 reviews and CI is green
4. Configure it to automerge PRs coming from the CI that is updating the Go version and these two: go.mod and go.sum

*** Alternative**: There is also Mergify that is worth checking.*

*** Alternative 2**: Specifically for automerging there is also this GH Action: https://github.com/marketplace/actions/merge-pull-requests*

*** Alternative 3**: There is also probot-auto-merge*

CI/CD - Go mod tidy

Description

Whenever there is a new version of the 3rd party dependencies used by the project (namely whatever is inside go.mod and go.sum) then let the project update itself without developer's intervention.

Test the first way

https://github.com/marketplace/actions/go-mod-tidy-pr-go-1-13-4

Test the second way

1. Learn about dependabot which is available also at GitHub Marketplace here
2. Write a GH Action workflow doing the following:
3. runs on push for the master branch`
4. gets triggered when go.mod, go.sum or the .github/workflow/<name>.yml has been modified.
5. fetches the project using actions/checkout
6. sets up Go 1.15 via actions/setup-go
7. After updating those files, it prepares a PR from go-mod-tidy branch, that gets automatically thanks to the automerge labels. For creating the PR as part of the same GH action use this, with the following settings:
8. Create a PR using actions/create-pull-request

```
name: Create Pull Request
uses: peter-evans/create-pull-request@v2
with:
  token: ${{ secrets.GITHUB_TOKEN }}
  commit-message: "chore(deps): go mod tidy"
  title: "chore(deps): go mod tidy"
  body: |
    Current `go.mod` and `go.sum` don't match the source code.
  branch: go-mod-tidy
  branch-suffix: timestamp
  labels: automerge
```

CI/CD - Create a Go Report card

Description

It's good to get a report for our code via a public service like goreport. Make sure it's visible on the front page, by adding the badge to the README

Acceptance Criteria

1. Create a card https://goreportcard.com/
2. Update README to reflect this as a badge

CI/CD - PR size estimation bot

Description

It's good to know how big (or not) it's a PR, so we can list them and see how many of those big one (or small ones) we have in the queue.

Acceptance Criteria

- Install and configure https://github.com/marketplace/pull-request-size

CI/CD - Protect the master branch

It is important that nobody can push directly to the master branch, so git history remains intact. The owner of the project would be always able to do it, although there is some hacks that disallow this:

- Read this nvie/gitflow#330
- Read this: https://gist.github.com/vlucas/8009a5edadf8d0ff7430
- Follow this: https://docs.github.com/en/github/administering-a-repository/enabling-required-status-checks
- Test it to make sure it works

CI/CD - Block Merging on WIP

Description

When a PR is not ready, developers signal these by putting WIP in the commit message, so nobody is reviewing.

Acceptance Criteria

- To ensure that this is not going to get merged by accident or we forget to remove the WIP from the title, use this https://github.com/marketplace/wip
- Test it to verify

CI/CD - PR author get auto-assigned

Description

In most cases, pull request author should be assigned an assignee of the pull request. This task makes sure the PR author assigns as an assignee (this is useful only if you use GitHub Projects and automations via each column).

Acceptance Criteria

- Install https://github.com/marketplace/actions/auto-author-assign
- Test it

CI/CD - Validate goreleaser changes

Description:

We need to make sure that if someone modifies.goreleaser.yml. It's a valid config and doesn't break. To do that we can run: goreleaser check

Acceptance Criteria:

1. Create a new GH Action (validate_goreleaser) that runs on push and pull request on the master branch
2. This should run on Ubuntu worker.
3. Fetch the code
4. Download goreleaser binary by running: curl -sfL https://install.goreleaser.com/github.com/goreleaser/goreleaser.sh | sh
5. and then it runs: goreleaser check

CI/CD - Reproducible Builds

Description

To make sure releases, checksums and signatures are reproducible, make some of the following modifications listed in https://goreleaser.com/customization/build/#reproducible-builds

Acceptance criteria

- Read and implement goreleaser reproducible builds configuration

CI/CD - Build multi-platform binaries

Description

Build binary/executables for Linux, Mac, Windows, *BSD for 32-bit, 64-bit, ARM-6 (that covers Raspberry Pi A, A+, B, B+, Zero), ARM-7(that is RPi2 and RPi3), ARM64 (that is RPi4).

Acceptance Criteria

- Use goreleaser to build all of those on the CI (GH Actions Workflow)
- Make sure they get published when there's a new release

CI/CD - Create .gitignore

Description

Debugging, running the make file, testing, doing all sorts of things, create some files locally. You don't want to commit any of those files. So please make sure they are ignored by git.

NOTE: This should be done by the end of the CI/CD tasks

Acceptance Criteria

1. Create gitignore file
2. Put inside all the files we don't need to be part of the repo.

CI/CD - Bot for conventional commits

Description

It is essential we all have the same guideline/principle for pushing commit messages. Having a bot to ensure that will slow down the speed of an unorganised team and it will speed up the organised one.

Acceptance Criteria

1. Read and understand https://www.conventionalcommits.org/en/v1.0.0/
2. Now create and configure this Semantic probot for the repo

CI/CD - Checksum and gpg signatures

Description

When there is a new release, lot's of artifacts will be published (executables, binaries, etc). Signing those ensures that have been generated by trusted members (maintainers) of the project, so the end-user can verify that by comparing the generated signature with my public signing key.

Acceptance Criteria

Read https://goreleaser.com/customization/sign/
Read https://goreleaser.com/customization/checksum/
Implement both of them as part of the .goreleaser.yml
Embed them into the GH Actions workflow when there's a new release

CI/CD - Convert TODO into an issue

Description

Sometimes devs use the keyword TODO in the source code in order to remind themselves to come back and fix that. Yeah, this never happens because they forget about it. Make sure they don't, by having a bot looking for that and creating a GH issue for it automatically.

Acceptance Criteria

1. Install https://github.com/marketplace/actions/todo-actions
2. Test it if it works and how it looks like. What labels can be applied, etc...

CI/CD - Build rpm, deb and apk packages

Description

Use NFPM to automatically build and release rpm (for RedHat based), deb (for Ubuntu/Debian based) and apk packages.

Acceptance criteria

1. Find out the appropriate configuration needed for the .goreleaser.yml
2. Find out the corresponding section we need to add to the Github Action workflow in order to publish those artifacts when a new release is made.

CI/CD - Mention contributors

Description

There are many ways to highlight your contributors. One of those is via the readme. There are plenty of ways to do it. Check other github repos do get inspired (e.g. this).

Acceptance Criteria

1. README highlights our contributors
2. If there is a new contributor is should be added automatically to that

CI/CD - mkdocs for documentation

Description

Create a static HTML generator using mkdocs to create the HTML files. Everytime there is a change to those markdown files, the CI will trigger a new build of the site.

Acceptance Criteria

1. Create a demo to check if this works
2. Put documentation in markdown into /docs
3. Use GH Actions
4. Create a GH Action workflow to build the new HTML files when there is a change in markdown
5. Check the changes reflect the website

CI/CD - Create templates for issues and PRs

Description

It’s a good practice to create a skeleton template for filling bugs or requesting for features.

Acceptance Criteria

1. Use https://docs.github.com/en/github/building-a-strong-community/about-issue-and-pull-request-templates
2. Test it

CI/CD - Add badges shield

Description

It's always good to have a nice overview of the project on the front page. This is usually done via the README and its badges. The most famous service for that is the shield. You can do that manually, but this time we will try an automation.

Acceptance Criteria

1. Install https://github.com/marketplace/actions/badge-it
2. Check if it interferes with already present badges

CI/CD - Document how to update Go

Description

The project is going to always follow the latest stable Go programming language. As a result, we need to manually bump it whenever the newer version gets available.

NOTE: This task assumes that all the CI/CD pipelines are setup along with the Documentation.

Acceptance Criteria

1. Find out all the necessary steps that need to be made to bump up to the new version of Go.
2. See if you can automate it with a script
3. Write a GH Action workflow that gets triggered manually and creates a PR with the changes
4. Make sure the PR gets automatically merged by a bot

IMPORTANT: Please look at https://github.com/marketplace/actions/ensure-latest-go in case this is already automated

CI/CD - Build scoop packages (Windows)

Description

Build and release scoop packages so Windows users can install romie like a breeze.

Acceptance Criteria

1. Use goreleaser scoop
2. Find out what configuration is needed for .goreleaser.yml
3. How to embed this to GH Actions workflow so new scoop packages are delivered when there's a new release

CI/CD - Build a docker container

Description

Build and release a docker container image and publish it automatically to docker.io. There should be two kinds of image tag: the latest and the vX.Y.Z that points to the specific version number.

Acceptance Criteria

1. Use goreleaser docker
2. Find out what config is needed for .goreleaser.yml
3. Write the dockerfile template
4. Embed this as part of the GH Actions workflow to get those published to dockerhub registry after a new release is made

CI/CD - Build snap packages

Description

Build and release snaps so any Linux user who uses snap, can install romie.

Acceptance Criteria

1. Use goreleaser snapcrafts
2. Find out what configuration is needed for .goreleaser.yml to achieve that.
3. How to embed this into GitHub Action workflow when there's a new release.

CI/CD - Build brew packages

Description

Use goreleaser brew to build and publish romie so MacOS users can install it easily.

Acceptance Criteria

1. Find out what needs to be done to get brew packages build automatically.
2. What configuration is needed for .goreleaser.yml
3. How to embed this into a Github actions workflow so those gets delivered upon a new release?

Baby steps
Write your own utility pkg

Write basic utilities

Examples:

- create directory
- create file
- move file
- sort
- implement progress bar while downloading a file
- fetch HTML objects
- other checks ... e.g. http://golangcookbook.com/

Why doing that?

- Everything that goes into internal/pkg
- Learn about importing and exporting packages
- Learn about documentation in Go (with examples)
- Learn about testing and mocking

You can use them for every Go project.

Write basic utilities - list files in a directory

Write a function/method lists files for a given directory (equivalent of Linux command **ls**)

Example: content, err := list(dir)

Acceptance Criteria

1. Make sure it works cross-platform
2. Make sure the directory exists and you have read permissions to it.
3. Put the function into internal/utils package
4. For logging use https://github.com/Sirupsen/logrus
5. Write a unit test
6. Make sure there are testable examples

Hints:

- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help:

- https://www.youtube.com/watch?v=ifBUflb7kdo
- For logging https://linuxhint.com/golang-logrus-package/

Write basic utilities - Check internet connectivity

Write a function checking if the host has the ability to communicate with the internet. For example try to ping to 8.8.8.8 and see if you get reply.

Example:

```
if internetOK {

      // do something

}
```

Acceptance Criteria

1. Write the function as part of the internal/utils package
2. Write unit test
3. In case of specific errors please store them separately to internal/errors pkg

Extra Help:

- click here

Write basic utilities - Check if folder exists

Write a function that checks if a specific directory exists on disk

Example:

```
if folderExists(directory) {
  // do something
}
```

Acceptance Criteria

1. Use https://github.com/spf13/afero for filesystem abstraction
2. Make sure it works cross-platform
3. For logging use https://github.com/Sirupsen/logrus
4. Put the function into internal/utils package
5. Write a unit test
6. Make sure there are testable examples

Hints:

- Writing a unit test for filesystem checking is not trivial. You should NOT create a real file on the system, because then your test will depend on the I/O of the file system itself. The last resort is mocking the filesystem. There are quite a few powerful libraries like spf13/afero for this purpose (mocking of a filesystem). These packages will create temporary files in the background and clean up afterward.
- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help:

- https://golangcode.com/check-if-a-file-exists/
- https://www.youtube.com/watch?v=ifBUflb7kdo
- https://github.com/spf13/afero#using-afero-for-testing
- An example using Afero for testing: http://krsacme.com/golang-fs-tests/
- For logging https://linuxhint.com/golang-logrus-package/

NOTICE: If you are unable to mock this test, in worst case scenario, create a folder internal/utils/testdata and put actual files you can use for your test.

Write basic utilities - Check if file exists

Write a function that checks if a specific file exists on disk

Example:

```
if fileExists(filename) {
  // do something
}
```

Acceptance Criteria

1. Use https://github.com/spf13/afero for filesystem abstraction
2. Make sure it works cross-platform
3. Put the function into internal/utils package
4. For logging use https://github.com/Sirupsen/logrus
5. Write a unit test
6. Make sure there are testable examples

Hints:

- Writing a unit test for filesystem checking is not trivial. You should NOT create a real file on the system, because then your test will depend on the I/O of the file system itself. The last resort is mocking the filesystem. There are quite a few powerful libraries like spf13/afero for this purpose (mocking of a filesystem). These packages will create temporary files in the background and clean up afterward.
- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help:

- https://golangcode.com/check-if-a-file-exists/
- https://www.youtube.com/watch?v=ifBUflb7kdo
- https://github.com/spf13/afero#using-afero-for-testing
- An example using Afero for testing: http://krsacme.com/golang-fs-tests/
- For logging https://linuxhint.com/golang-logrus-package/

NOTICE: If you are unable to mock this test, in worst case scenario, create a folder internal/utils/testdata and put actual files you can use for your test.

Write basic utilities - Check if ENV var exists

Write a function/method checks if an environment variable exists and returns its.

Example:

```
homeDir, err := enVar("HOME")
```

Acceptance Criteria

1. Use https://github.com/spf13/afero for filesystem abstraction
2. Make sure it works cross-platform
3. Put the function into internal/utils package
4. Make sure it's crossplatform
5. For logging use https://github.com/Sirupsen/logrus
6. Write a unit test
7. Make sure there are testable examples

Hints:

- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help:

- https://gobyexample.com/environment-variables
- For logging https://linuxhint.com/golang-logrus-package/

NOTICE: If you are unable to mock this test, in worst case scenario, create a folder internal/utils/testdata and put actual files you can use for your test.

Write basic utilities - Check if value exists in array

Write a function checking if a specific value exists in an array and return it's index.

Example:

```
index, exists := isArrayContain("alpha")
if exists {
    // log index
}
```

Acceptance Test

1. Make the code part of internal/utils pkg
2. For logging use https://github.com/Sirupsen/logrus
3. Write a unit test
4. Make sure there are testable examples
5. For specific errors please put them into internal/errors pkg

Hints:

- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help

- https://github.com/drpaneas/gophergr/blob/master/content/post/basic_unit_testing.md
- For logging https://linuxhint.com/golang-logrus-package/

Write basic utilities - Delete file

Write a function/method that deletes a file (equiv to Linux command rm)

Example:

```
err := rm(filepath)
```

Acceptance Criteria

1. Make sure it works cross-platform
2. Put the function into internal/utils package
3. For logging use https://github.com/Sirupsen/logrus
4. Write a unit test
5. Make sure there are testable examples

Hints:

- Writing a unit test for filesystem checking is not trivial. You should NOT create a real file on the system, because then your test will depend on the I/O of the file system itself. The last resort is mocking the filesystem. There are quite a few powerful libraries like spf13/afero for this purpose (mocking of a filesystem). These packages will create temporary files in the background and clean up afterward.
- As an exercise, try to write the test first. Let it fail. Then try to fix it, by writing the actual implementation of the function.

Extra Help:

- https://golangcode.com/check-if-a-file-exists/
- https://www.youtube.com/watch?v=ifBUfIb7kdo
- https://github.com/spf13/afero#using-afero-for-testing
- An example using Afero for testing: http://krsacme.com/golang-fs-tests/
- For logging https://linuxhint.com/golang-logrus-package/

NOTICE: If you are unable to mock this test, in worst case scenario, create a folder internal/utils/testdata and put actual files you can use for your test.

Learn the ecosystem

Learn about specific topics

- URL shortener
- HTML Link Parser
- Sitemap Builder
- CLI Task Manager
- FIle Renaming Tool
- Twitter Retweet CLI
- Build Images
- and others ...

- https://exercism.io/my/tracks/go
- https://gophercises.com/
- https://spf13.com/presentation/building-an-awesome-cli-app-in-go-oscon/
- https://astaxie.gitbooks.io/build-web-application-with-golang/content/en/
- https://www.devdungeon.com/content/web-scraping-go#make_http_get_request
- https://algorithmswithgo.com/?referer=gotime
- https://dave.cheney.net/high-performance-go-workshop/gopherchina-2019.html
-

Start writing unit-tests

Use simple go

- go test -v ./...
- naming conventions
- *T testing pointer
- log vs error vs fatal
- coverage

Use simple go: Generate table driven tests

```go
func TestAddTwoNumbers(t *testing.T) {
    x := 3
    y := 5
    t.Logf(format:"Testing: %v + %v", x, y)
    result := AddTwoNumbers(x,y)
    if result != 8 {
        t.Fatalf(format:"Expected %v, but we got %v", x+y, result)
    }
```

```go
// AddTwoNumbers returns the sum of X and Y (integers)
func AddTwoNumbers(x , y int) int {
    return x + y
}
```

```go
func TestAddTwoNumbers(t *testing.T) {
    type args struct {
        x int
        y int
    }
    tests := []struct {
        name string
        args args
        want int
    }{
        // TODO: Add test cases.
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            if got := AddTwoNumbers(tt.args.x, tt.args.y); got != tt.want {
                t.Errorf("AddTwoNumbers() = #{got}, want #{tt.want}")
            }
        })
    }
}
```

Use an external testing library

- https://github.com/stretchr/testify
  - if you are into “assert”
- BDD:
  - https://github.com/onsi/ginkgo or https://github.com/cucumber/godog
  - if you are used to “cucumber” or given, when, then
- GoMock:
  - https://github.com/golang/mock
- Filesystem mocking
  - https://github.com/spf13/afero
- Any other specific test library for your use-case (e.g. rest api)

Structure your project

- Read https://github.com/golang-standards/project-layout
- Use Proper Licence
- Have a README.md
  - Docs for users
  - Docs for devs
- Have a CONTRIBUTING.md

Start contributing
and learn from other people

Find a project

- All Go repositories listed by number of stars
- Awesome-Go: https://github.com/avelino/awesome-go
- Trending: https://github.com/trending/go?since=daily

- Kubernetes (minikube, helm, controllers, kube-builder, mixin, ingress-nginx, others ...)
- Prometheus
- grafana/Loki
- sirupsen/logrus
- etcd
- caddy
- grafana/loki
- AWS SDK Go
- Traefik
- Istio (Service Mesh)
- Hugo
- Gin-gonic
- syncthing
- go-ethereum
- hashicorp/{packer, terraform, vault}
- spf13/cobra & viper
- k3s
- openshift
- gocolly/colly
- ipfs/go-ipfs
- matrix-org/dendrite

Improve the existing codebase

- Write tests
- Improve already written tests
- Add any CI gimmick from the previous slides
- Review PRs related to tests and CI

- Start refactoring
  - Use Design Patterns: https://refactoring.guru/design-patterns/go

- Fix bugs
- Implement new feature

Thanks for your time