在 GitHub 上构建一个看上去正规的 Golang 项目

郭旭东x 2019-07-22 4145浏览量

简介： 接触 golang 时间很长，但是真正动手开始写 golang 也就是在最近。跟着我在 GitHub 上构建一个看上去正规的 Golang 项目。

前言

接触 golang 时间很长，但是真正动手开始写 golang 也就是在最近。虽然写的不多，但是见过的 golang 项目可是不计其数，从 Kubernetes 和 istio 到亲身参与的 kustomize 再到 Kubernetes 生态圈的众多小工具，比如： kubeval 、 kubedog 等。从项目使用者和贡献者的角度接触了各种形形色色的 golang 项目。作为一个开发人员，在享受各种开源项目带来便利的同时，也希望自己动手开发一个 golang 项目。以我阅项目无数的经验，那么肯定要构建一个看上去正规的 GitHub 项目。

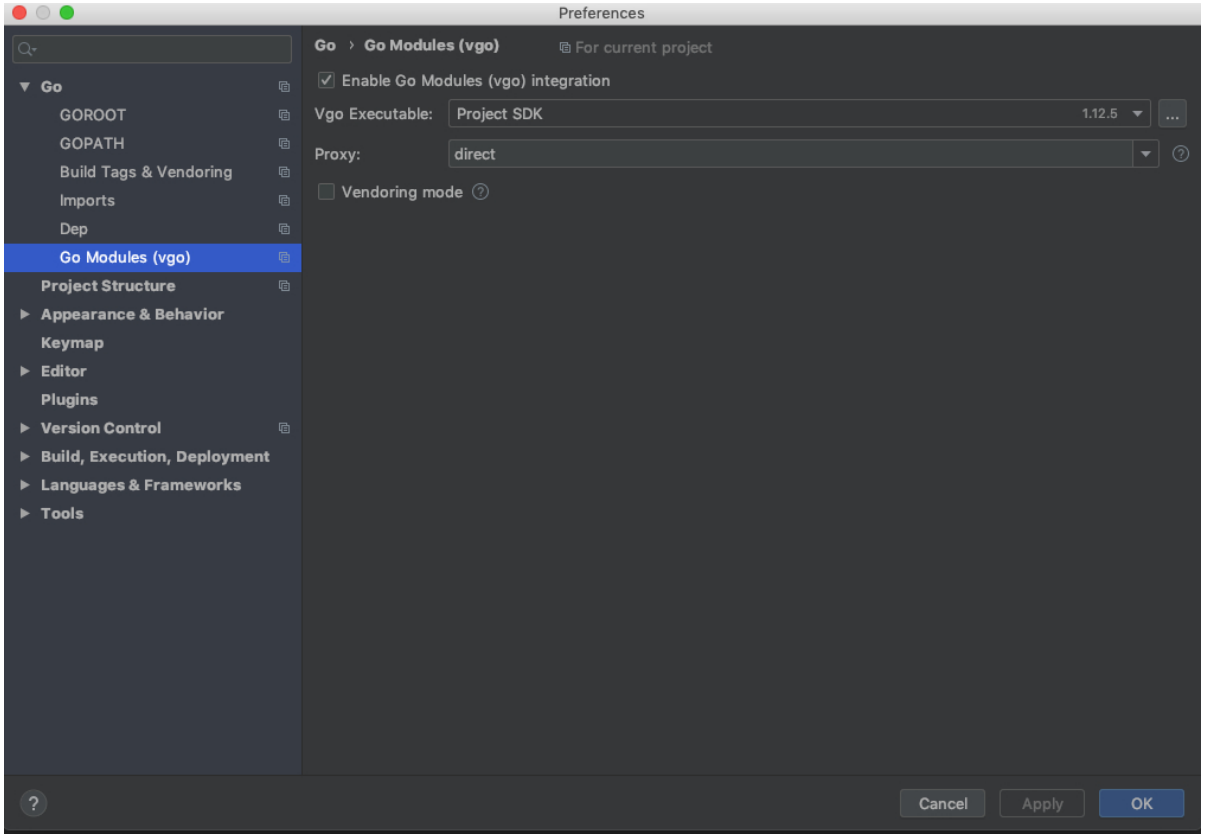
GoLand 设置

Go 开发环境的安装网上教程很多，这里就不做介绍了。这里主要介绍一下在 GoLand 上开发环境的设置，这里的设置主要在 MacOS 上进行，其他系统可能有所不同。

使用Goland IDE vgo

vgo 是基于 Go Module 规范的包管理工具，同官方的 go mod 命令工具类似。

1. 开启 `vgo` ， `GoLand -> Preferences -> GO -> Go Modules(vgo)`



1. 手动修改 `go.mod`

   其中 latest 为最新版本，GoLand 会去下载最新依赖代码，下载成功后会修改 `go.mod` 并且生成 `go.sum` 依赖分析文件。

```
module github.com/sunny0826/hamal

go 1.12

require (
    github.com/mitchellh/go-homedir latest
    github.com/spf13/cobra latest
    github.com/spf13/viper latest
)
```

2. 更新成功

   在更新成功后，会生成 `go.sum` 文件并修改 `go.mod` 文件。

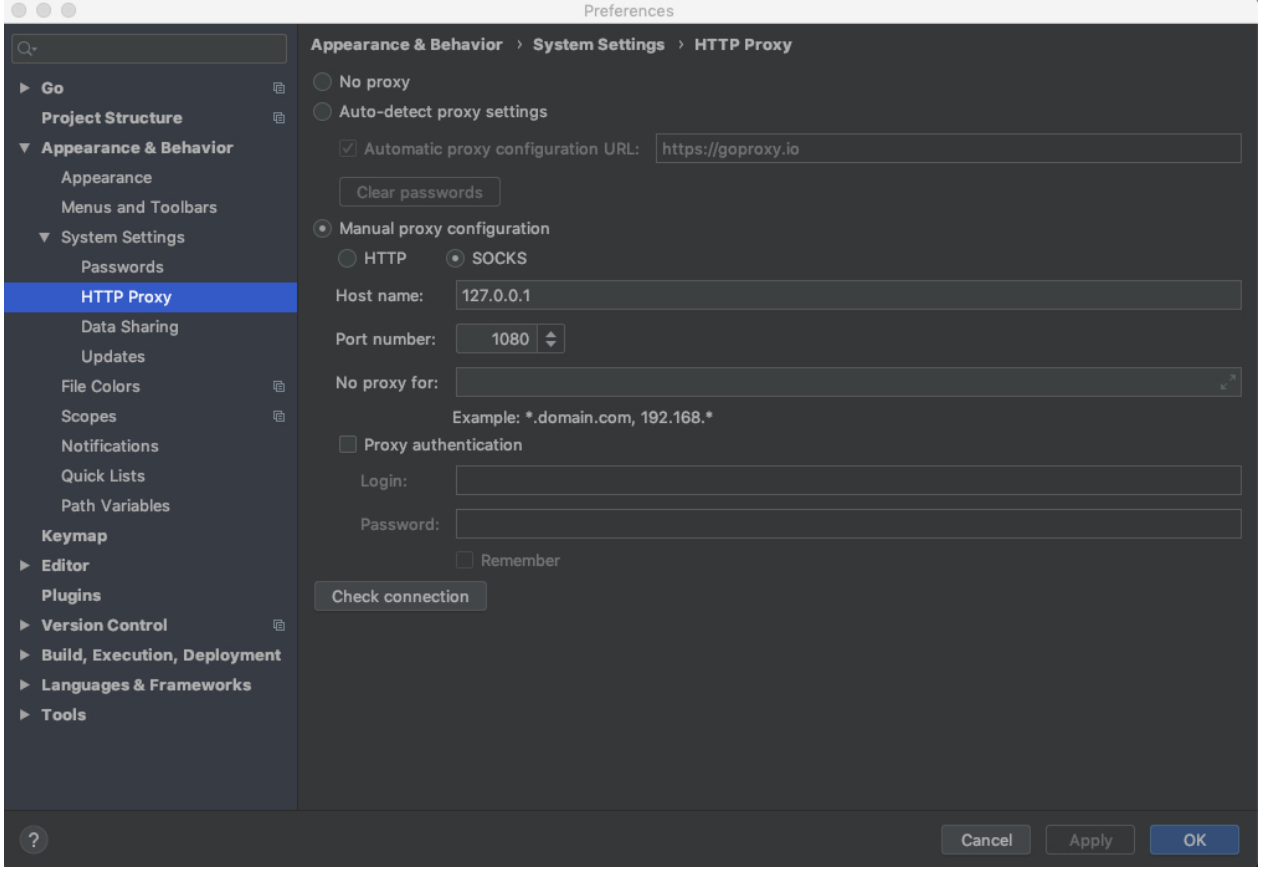
```
module github.com/sunny0826/hamal

go 1.12

require (
    github.com/mitchellh/go-homedir v1.1.0
    github.com/spf13/cobra v0.0.5
    github.com/spf13/viper v1.4.0
)
```

3. 使用快捷键 `⌥(option)+↩(return)` 或者点击鼠标右键, 选择 `Sync packages of github.com/sunny0826/hamal` 在 import 处导入依赖。

配置代理

如果要选出 golang 最劝退一个原因，那么依赖下载难肯定得票最高！这个时候一个合适的梯子就很重要了，如果没有这个梯子，上面的这步就完全无法完成。这里主要介绍 GoLand 上的配置，Shadowsocks 的安装和配置就不做介绍了。

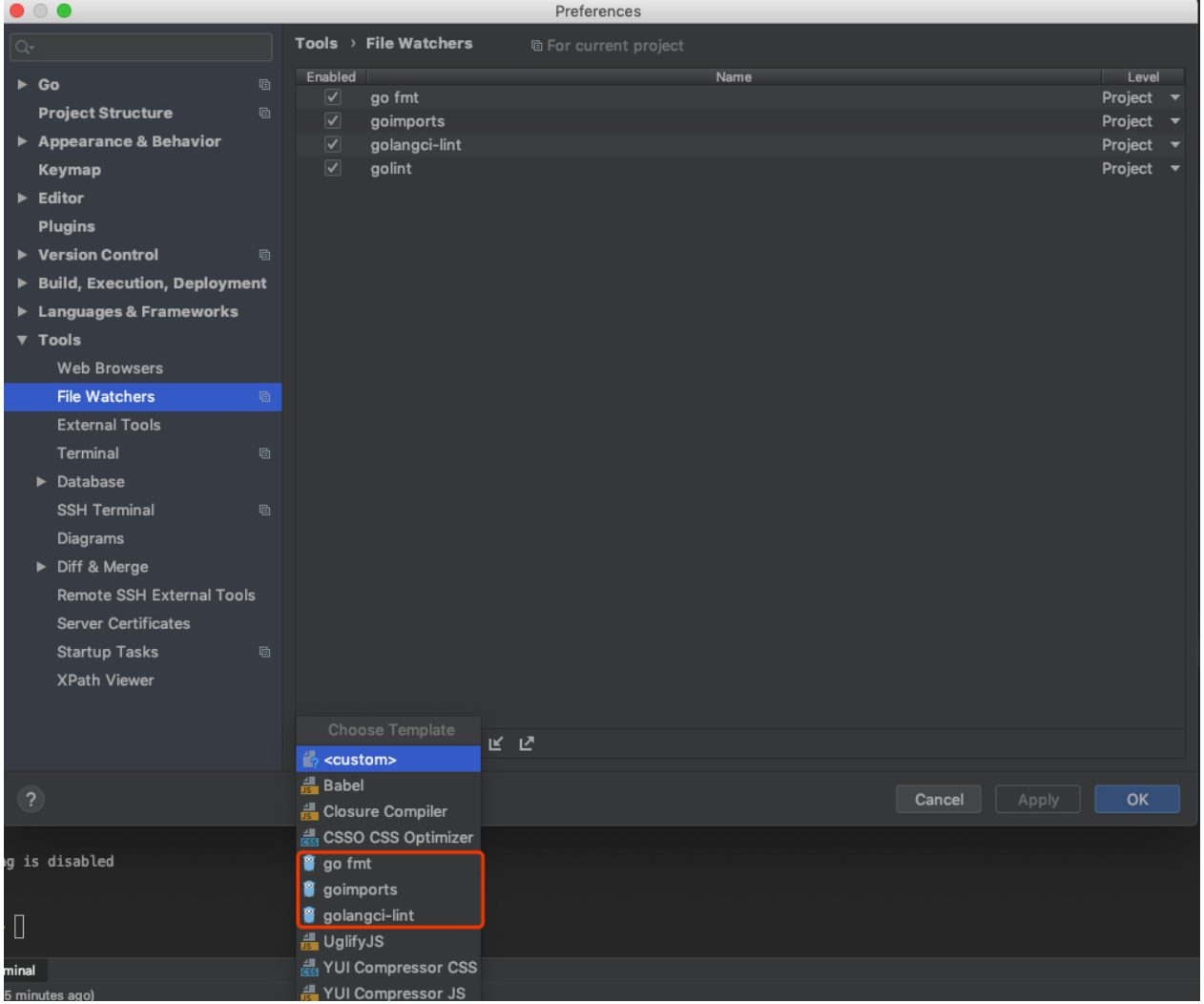
`GoLand -> Preferences -> Appearance & Behavior -> System Settings -> HTTP Proxy` 这里设置好之后，别忘了点击 `Check connection` 测试一下梯子搭成没有。



配置 go fmt 、 goimports 和 golangci-lint

这三个工具都是 GoLand 自带的，设置起来十分简单: `GoLand -> Preferences -> Tools -> File Watchers` ，点击添加即可。之后在写完代码之后就会自动触发这3个工具的自动检测，工具作用：

- `go fmt` ：统一的代码格式化工具。
- `golangci-lint` ：静态代码质量检测工具，用于包的质量分析。
- `goimports` ：自动 import 依赖包工具。



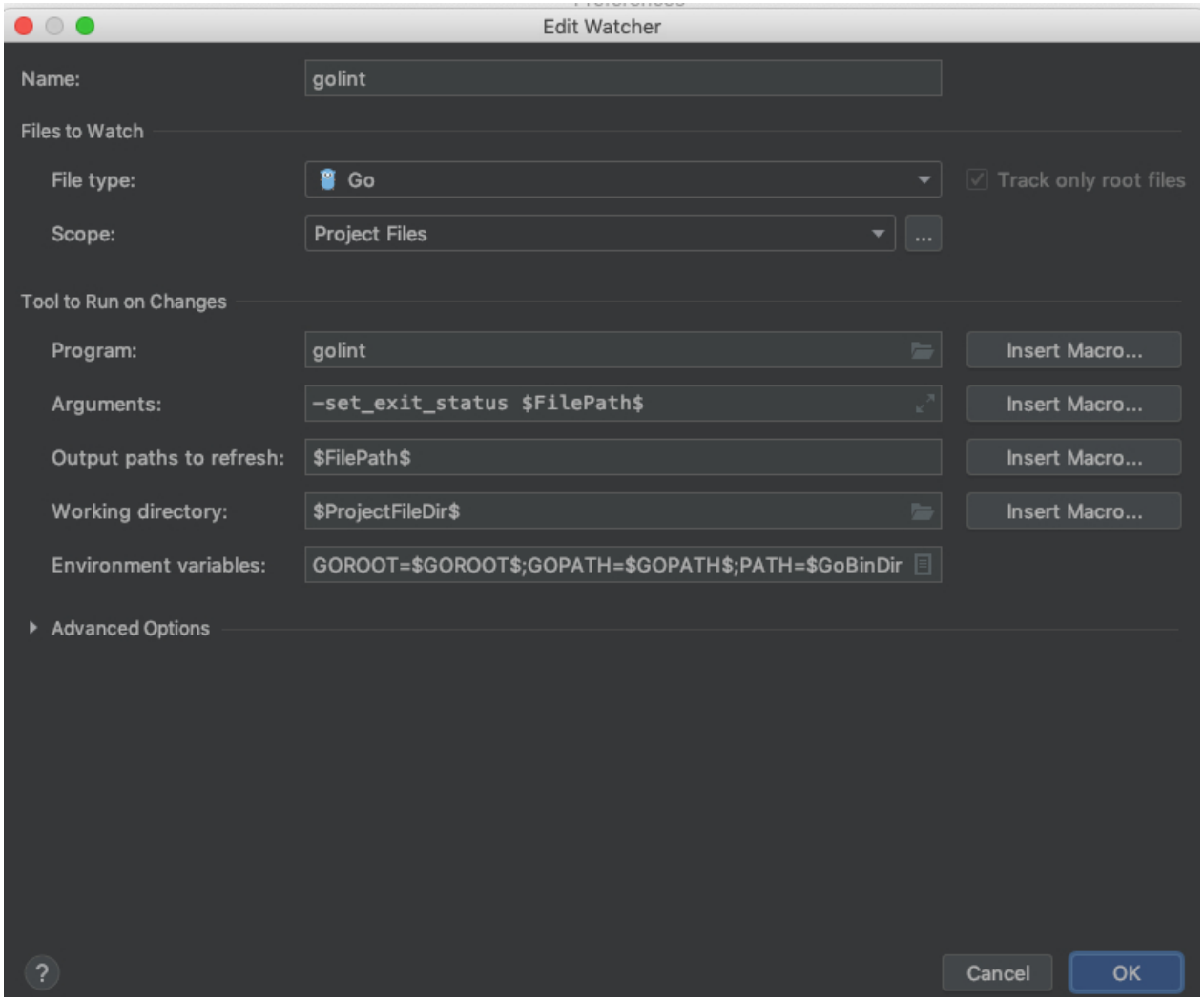
安装配置 golint

GoLand 没有自带 `golint` 工具，需要手动安装：

```
mkdir -p $GOPATH/src/golang.org/x/
cd $GOPATH/src/golang.org/x/
git clone https://github.com/golang/lint.git
git clone https://github.com/golang/tools.git
cd $GOPATH/src/golang.org/x/lint/golint
go install
```

安装成功之后将会在 `$GOPATH/bin` 目录下看到自动生成了 `golint` 二进制工具文件。

GoLand 配置 `golint`，修改 `Name`，`Program`，`Arguments` 三项配置，其中 `Arguments` 需要加上 `-set_exit_status` 参数，如图所示：



Travis CI 持续集成

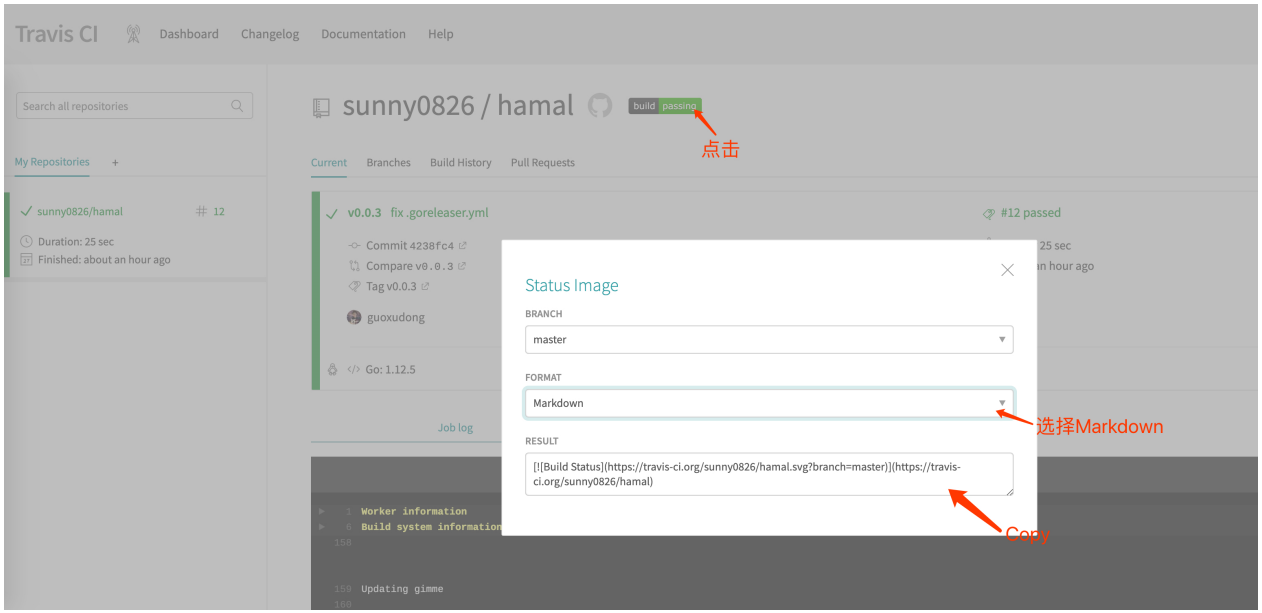
在 Github 上装逼怎么能少的了 Travis CI，直接登录 Travis CI，使用 GitHub 登录，然后选择需要使用 Travis CI 的项目，在项目根目录添加 `.travis.yml`，内容如下：

```
language: go

go:
  - 1.12.5
```

```
sudo: required

install:
  - echo "install"

script:
  - echo "script"
```

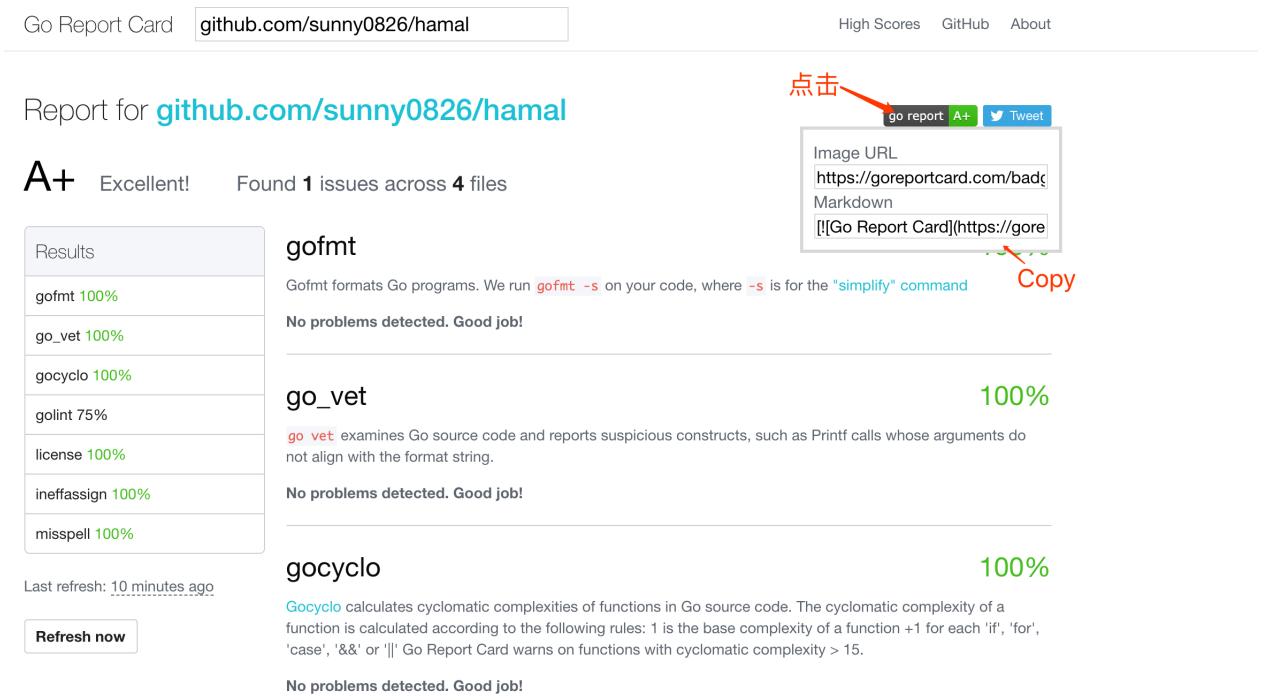
这里只是一个示例，在每次 push 代码之后，都会触发 CI，具体语法可以参看官方文档。

重点： 你以为使用 Travis CI 就是为了持续集成吗？那就太天真了！使用 Travis CI 当然为了他的 Badges，将 `RESULT` 拷贝到你的 `README.md` 里面就好了。



GO Report Card

__又一重点__：我们在 GoLand 上安装了 `golint` 等工具进行代码质量检测，在撸码的时候就能进行代码检查，那么这个就是为了纯装逼了。GO Report Card 是一个 golang 代码检测网站，你只需把 Github 地址填上去即可。获取 Badges 的方法和 Travis CI 类似，将 MarkDown 中的内容拷贝到 `RERADME.md` 中就好。



GoReleaser

持续集成有了，代码检查也有了，再下面就是怎么发布一个漂亮的 release 了。如果还在手动发布 release ，那么就又掉 low 了。使用 GoReleaser 一行命令来发布一个漂亮的 release 吧。

由于使用的的 MacOS ，这里使用 `brew` 来安装：

```
brew install goreleaser
```

在项目根目录生成 `.goreleaser.yml` 配置：

```
goreleaser init
```

配置好了以后要记得往 `.gitignore` 加上 `dist` ，因为 goreleaser 会默认把编译编译好的文件输出到 `dist` 目录中。

goreleaser 配置好后，可以先编译测试一下：

```
goreleaser --skip-validate --skip-publish --snapshot
```

注意： 首次使用 goreleaser 要配置 GITHUB_TOKEN，可以在这里申请，申请好之后运行下面的命令配置 `GITHUB_TOKEN`

```
export GITHUB_TOKEN=<YOUR_TOKEN>
```

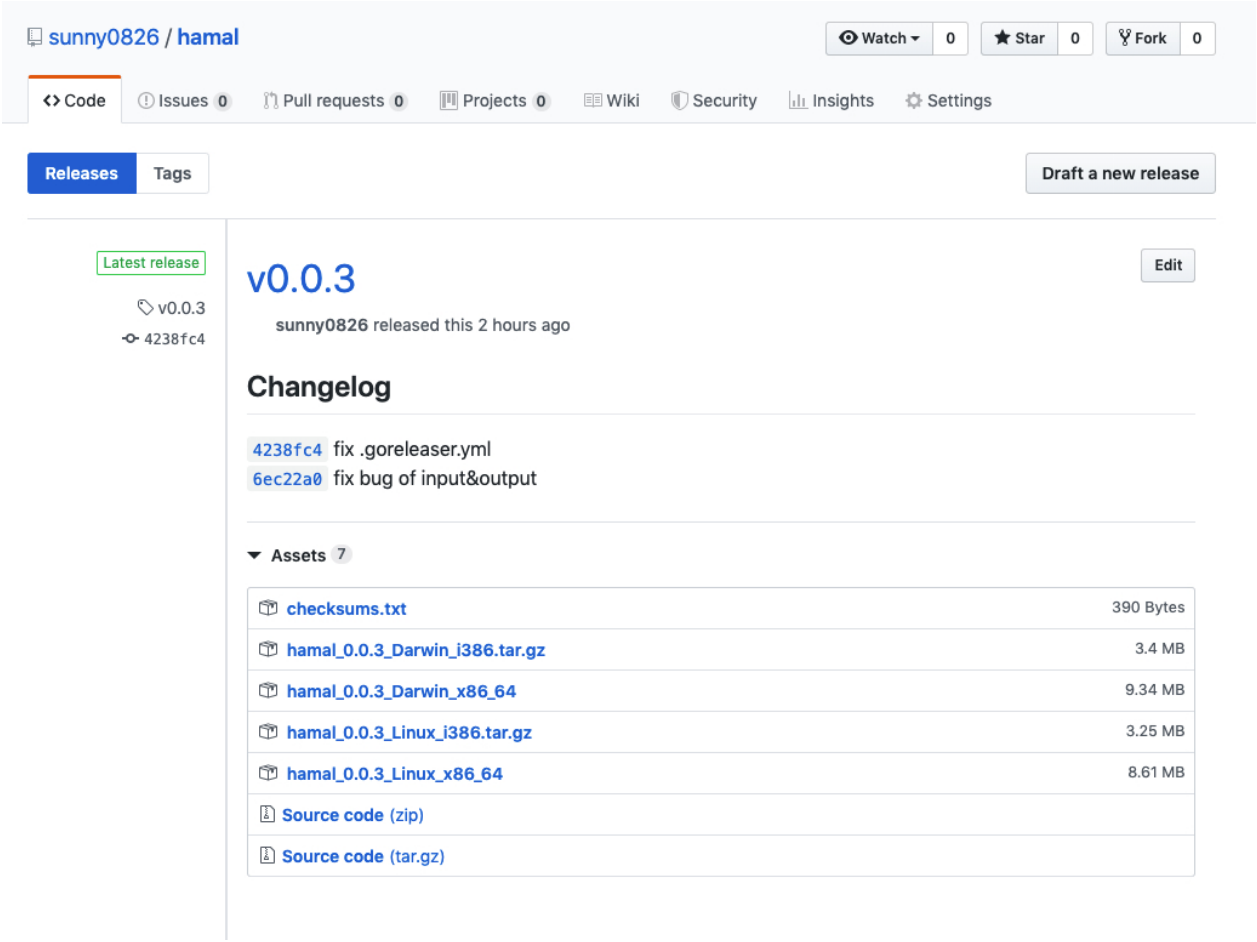
确保没有问题，那么就可以操作 git 和 goreleaser 来发布 release 了。

```
git add .
git commit -m "add goreleaser"
git tag -a v0.0.3 -m "First release"
git push origin master
git push origin v0.0.3
```

全部搞定后，一行命令起飞：

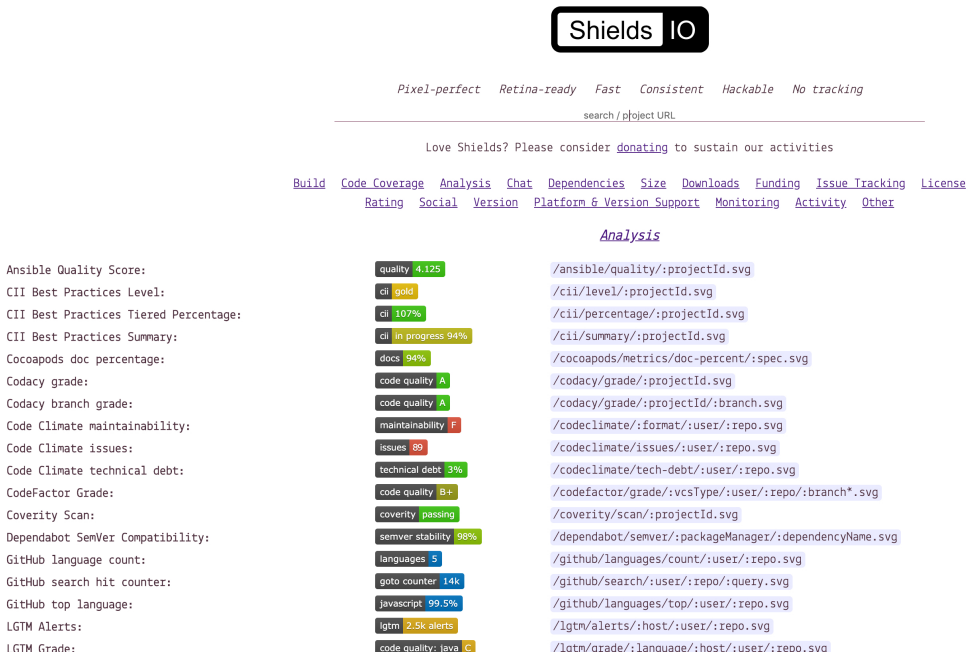
```
goreleaser
```

`goreleaser` 配合 CI 食用，效果更佳，这里就不做介绍了。



Badges 展示神器

这里介绍一个展示 Badges 的神器：https://shields.io/ 。这个网站提供各种各样的 Badges ，如果你愿意，完全可以把你的 GitHub README.md 填满，有兴趣的同学可以自取。



后记

到这里可以在 GitHub 上装逼的 golang 配置已经介绍的差不多了，其实还有 Codecov、CircleCI 等工具，这里就不做介绍了。这里要介绍的是我们的第一个 golang 项目 Hamal，该项目是一个命令行工具，用来在不同的镜像仓库之间同步镜像。由于我司推行混合云，使用了阿里云与华为云，而在阿里云或华为云环境互相推镜像的时候时间都比较长，所以开发这个小工具用于在办公网络镜像同步，同时也可以用来将我在 dockerhub 上托管的镜像同步到我们的私有仓库，欢迎拍砖。

Go 开发工具 git 持续交付

分享：

评论

登录后可评论