Introduction

There are many times in which you may install Go (either from source or from a binary distribution) and things don't work quite right. This page is meant to collect some common wisdom about problems that are relatively common or difficult to diagnose and provide tips and solutions.

Contents

Tips

Environment

To start out with, check the following first:

- GOROOT
  - This should *only* be set if you used a binary distribution and it's not installed in the default location.
- GOPATH
  - This should be set to the directory under which you want your source (and third party packages).
  - This can also be set to a list of absolute paths separated by : (or ; on Windows).
  - Note that ~/some/path is not absolute and will probably not work the way you expect (try $HOME/some/path instead).
  - GOPATH should not be set to or contain GOROOT
- GOBIN
  - This should only be set if you *really* know what you're doing... The default should be fine.
- GOOS, GOARCH, GOHOSTOS, GOHOSTARCH
  - You shouldn't need to set these in normal cases.

Under linux and darwin, make sure that any of the above variables which are set are actually exported. When you run the `env | grep GO` command, they should be listed. You can also check your environment with the `go env` command. In bash, this is done with the `export GOPATH` (if it's already set) or `export GOPATH=/path/to/gopath` command (similarly for the other variables), usually in your .bashrc or .bash_profile.

GOROOT vs GOPATH

Packages under GOROOT store their source files in

`$GOROOT/src/pkg/import/path/*.go`

Notice that this is `src/pkg` ; under GOPATH, source files are stored in

`$GOPATH/src/import/path/*.go`

Because of this inconsistency, it is generally not recommended that GOPATH be set to or contain GOROOT; its directories will be searched automatically for imports regardless of the GOPATH setting.

Troubleshooting

The `go build` command doesn't do anything!

The `go build` command will only produce a binary; if you run go build in a package directory, it will build the package normally (and report any compile errors), but it will not install it. For that, you use `go install`. If you think you're building a binary and none is produced, make sure you are in package `main` and that you do not have GOBIN set.

Why does `go get` report `"Fetching https://runtime/cgo?go-get=1"`?

If you have a source distribution, make sure that your packages are up-to-date. Also double check the environment above.

When cross compiling, I get `"runtime/extern.go:135: undefined: theGoos"`

Read [[WindowsCrossCompiling]] for some helpful scripts. You can also use the `--no-clean` argument when you're building the cross-compile toolchain via `make.bash`.

Why does `go get` work for some packages and report `permission denied` in `$GOROOT` for some others (with GOPATH set properly)?

If you at any point installed the package in `GOROOT` (either by having no `GOPATH` set or by including `GOROOT` itself in `GOPATH`) then there might still be a directory in `$GOROOT` (which is always checked first) that is overriding your `GOPATH`. To verify, run `go list -f {{.Dir}} importpath` and if it reports a directory under `$GOPATH` try deleting that first.

Still need help?

Visit us on IRC or ask on the mailing list. You will want to provide the output of the following commands, in addition to any errors you are getting:

Linux/darwin

```
go version
go env
env | grep GO
```

Windows

```
go version
go env
set | findstr GO
```