HW0-Environment Set Up

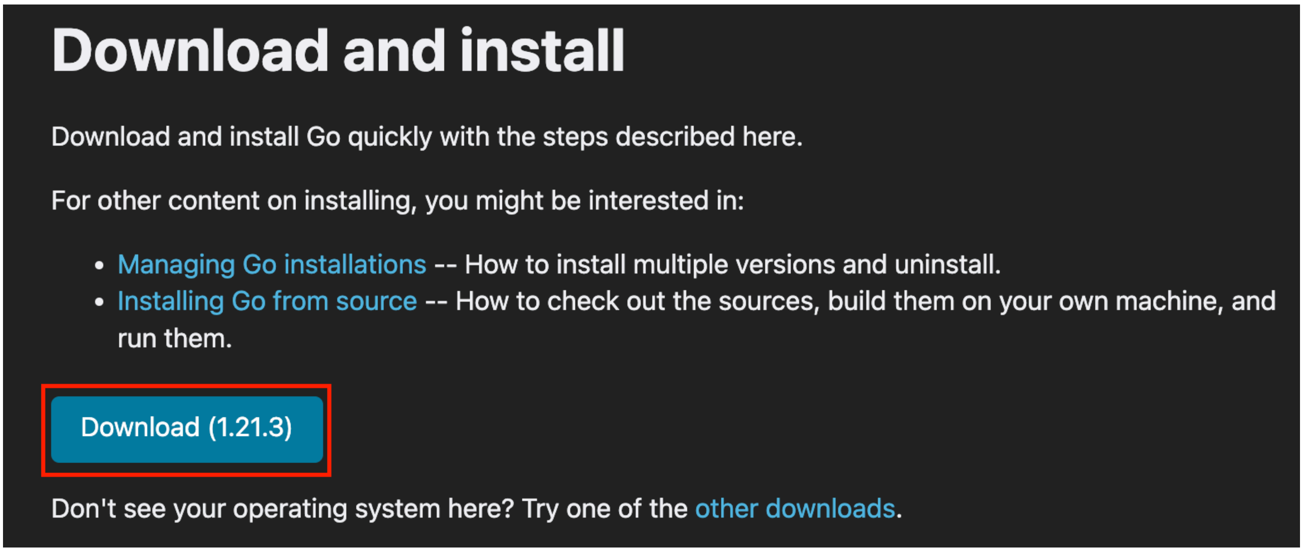
This assignment aims to guide you to learn how to install and set up the Golang environment. Please follow the following steps to build up your Golang environment.

Installation

You could refer to the official website to get the details of installation. Here we give the main steps to install Golang for different operating systems.

MacOS

Go to the official website and download the dmg package.



Download and install

Download and install Go quickly with the steps described here.

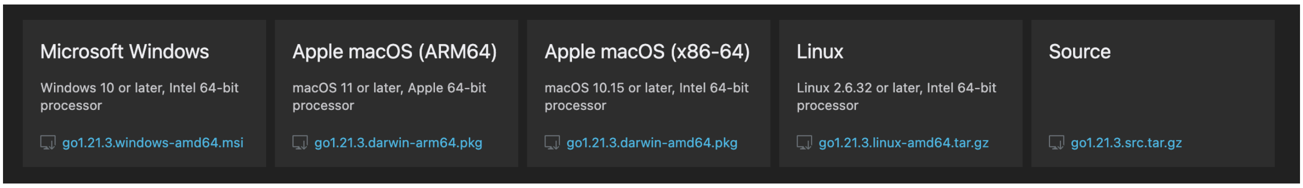
For other content on installing, you might be interested in:

- Managing Go installations -- How to install multiple versions and uninstall.
- Installing Go from source -- How to check out the sources, build them on your own machine, and run them.

Download (1.21.3)

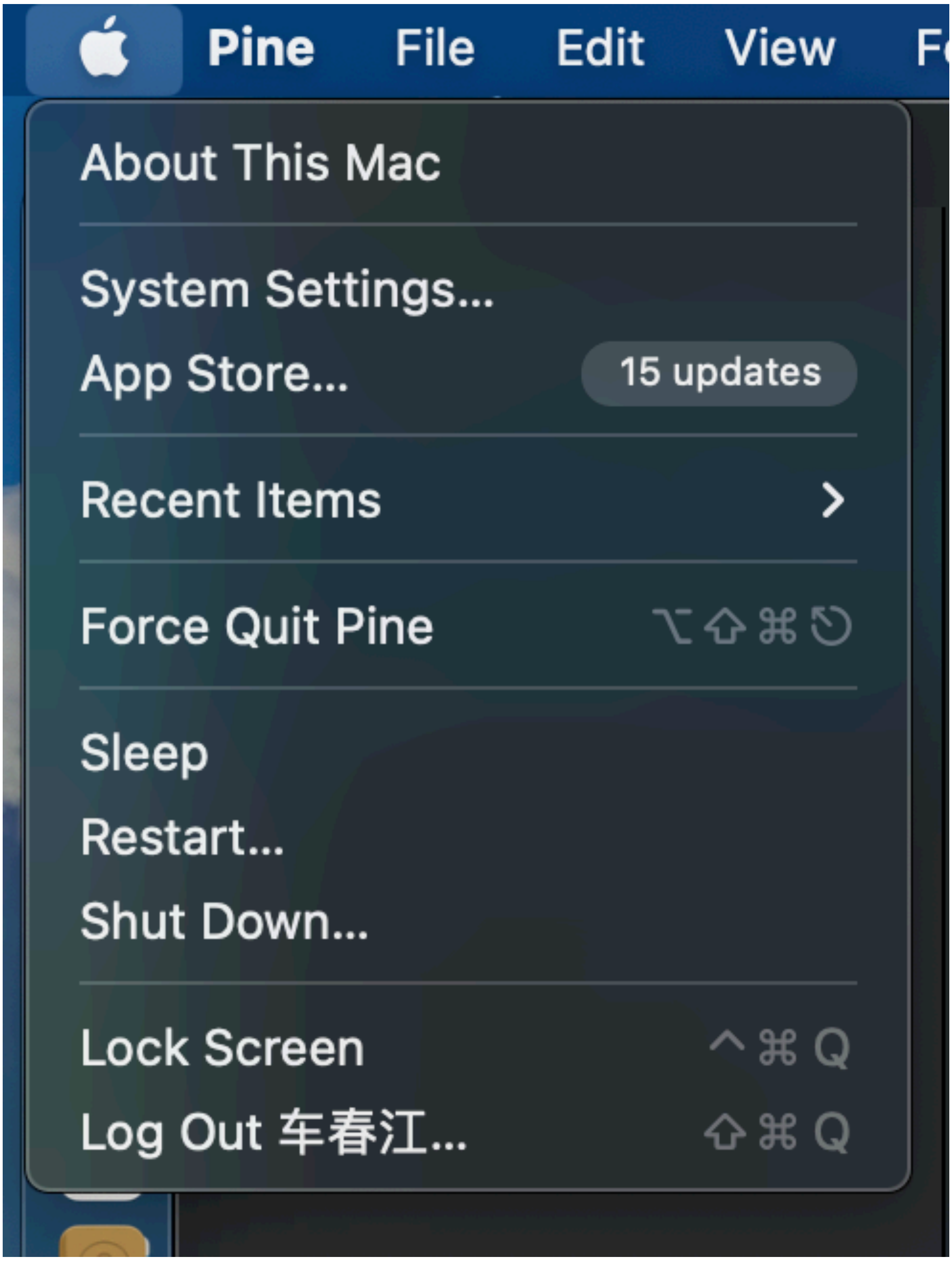
Don't see your operating system here? Try one of the other downloads.

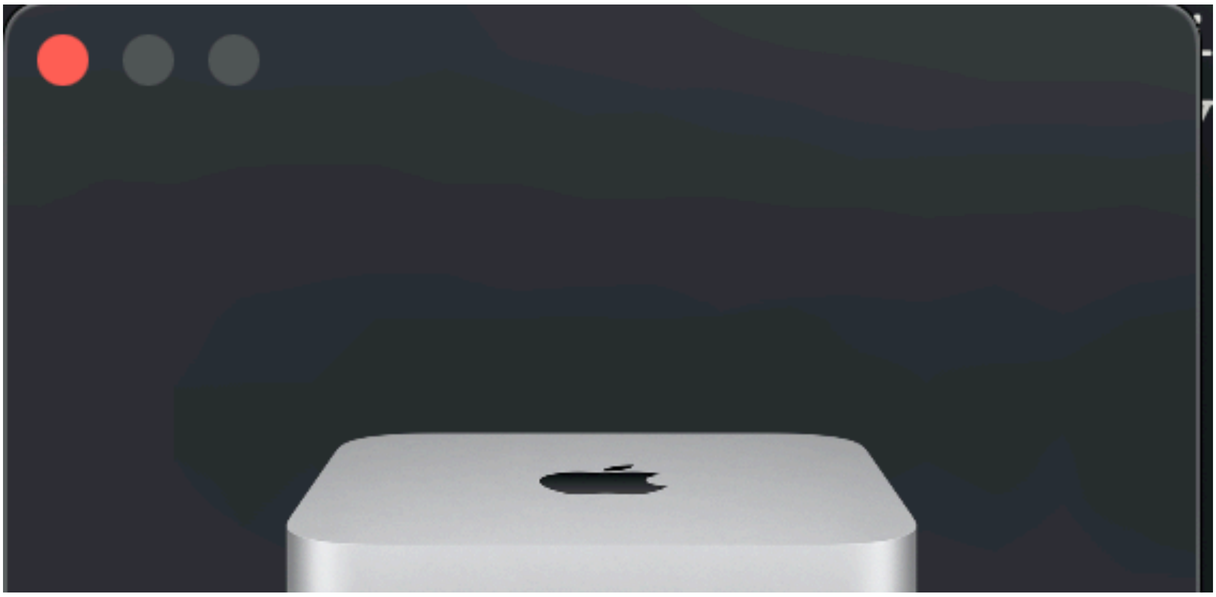
On the detailed page of installation, you are asked to select a version to download, like

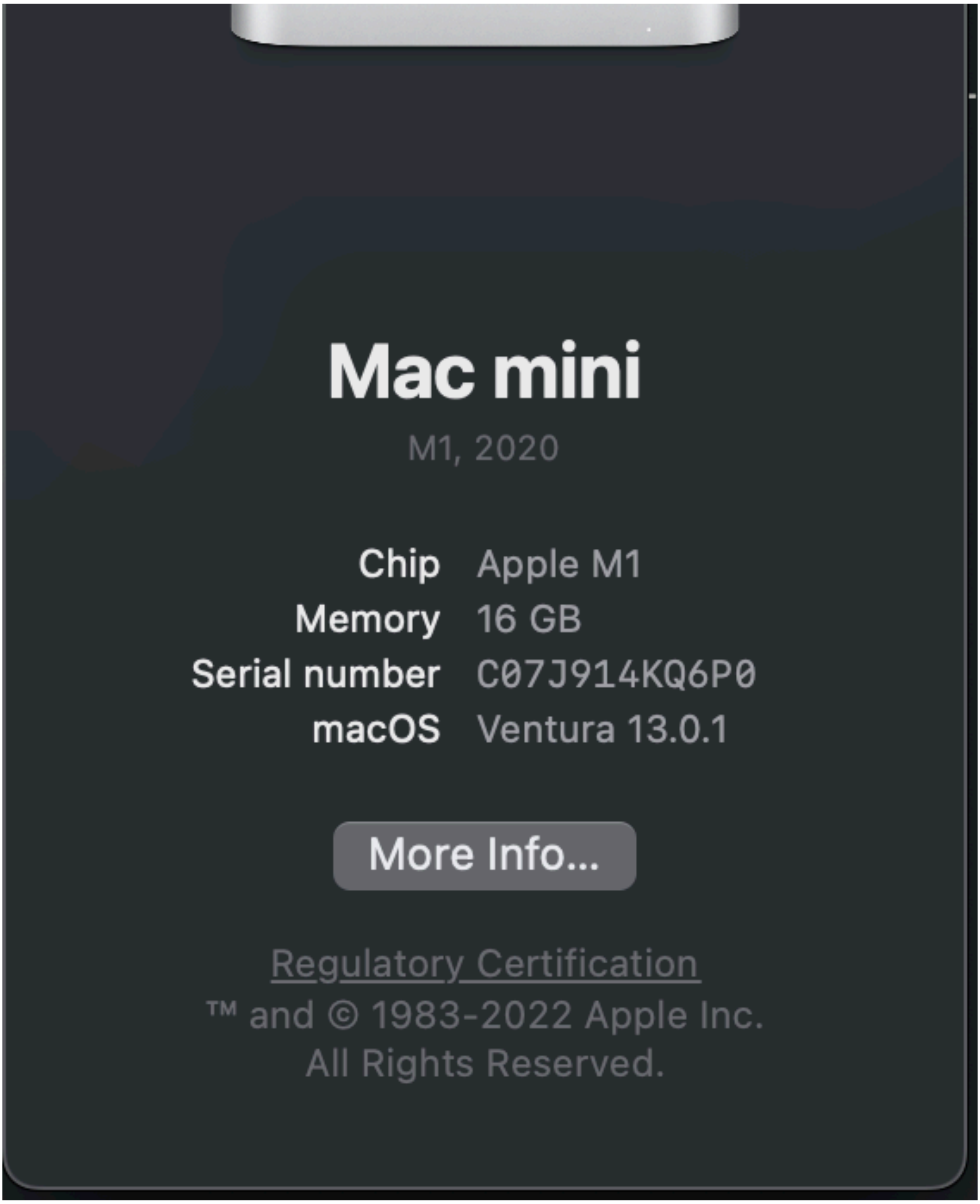


Microsoft Windows	Apple macOS (ARM64)	Apple macOS (x86-64)	Linux	Source
Windows 10 or later, Intel 64-bit processor	macOS 11 or later, Apple 64-bit processor	macOS 10.15 or later, Intel 64-bit processor	Linux 2.6.32 or later, Intel 64-bit processor	
go1.21.3.windows-amd64.msi	go1.21.3.darwin-arm64.pkg	go1.21.3.darwin-amd64.pkg	go1.21.3.linux-amd64.tar.gz	go1.21.3.src.tar.gz

If you are unsure what kind of version you should download, go to Apple Icon->About This Mac. You should be able to select the correct version according to the chip and OS version of your Mac. In the example shown as follows, the chip is Apple M1 and the OS version is 13.0.1, so the second one should be selected.


Pine
File
Edit
View
Fo
About This Mac
System Settings...
App Store...
15 updates
Recent Items
Force Quit Pine ⌥⇧⌘⎋
Sleep
Restart...
Shut Down...
Lock Screen ^⌘Q
Log Out 车春江... ⇧⌘Q





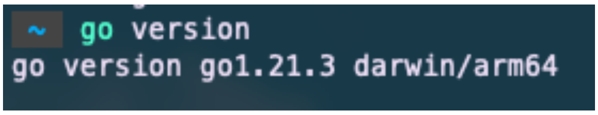
Mac mini

M1, 2020

Chip Apple M1
Memory 16 GB
Serial number C07J914KQ6P0
macOS Ventura 13.0.1

™ and © 1983-2022 Apple Inc.
All Rights Reserved.

The final step is clicking the downloaded dmg package and then completing the installation. You can verify whether the installation is successful or not by running `go version` in the Terminal. If it outputs the following log, then you successfully install Golang.



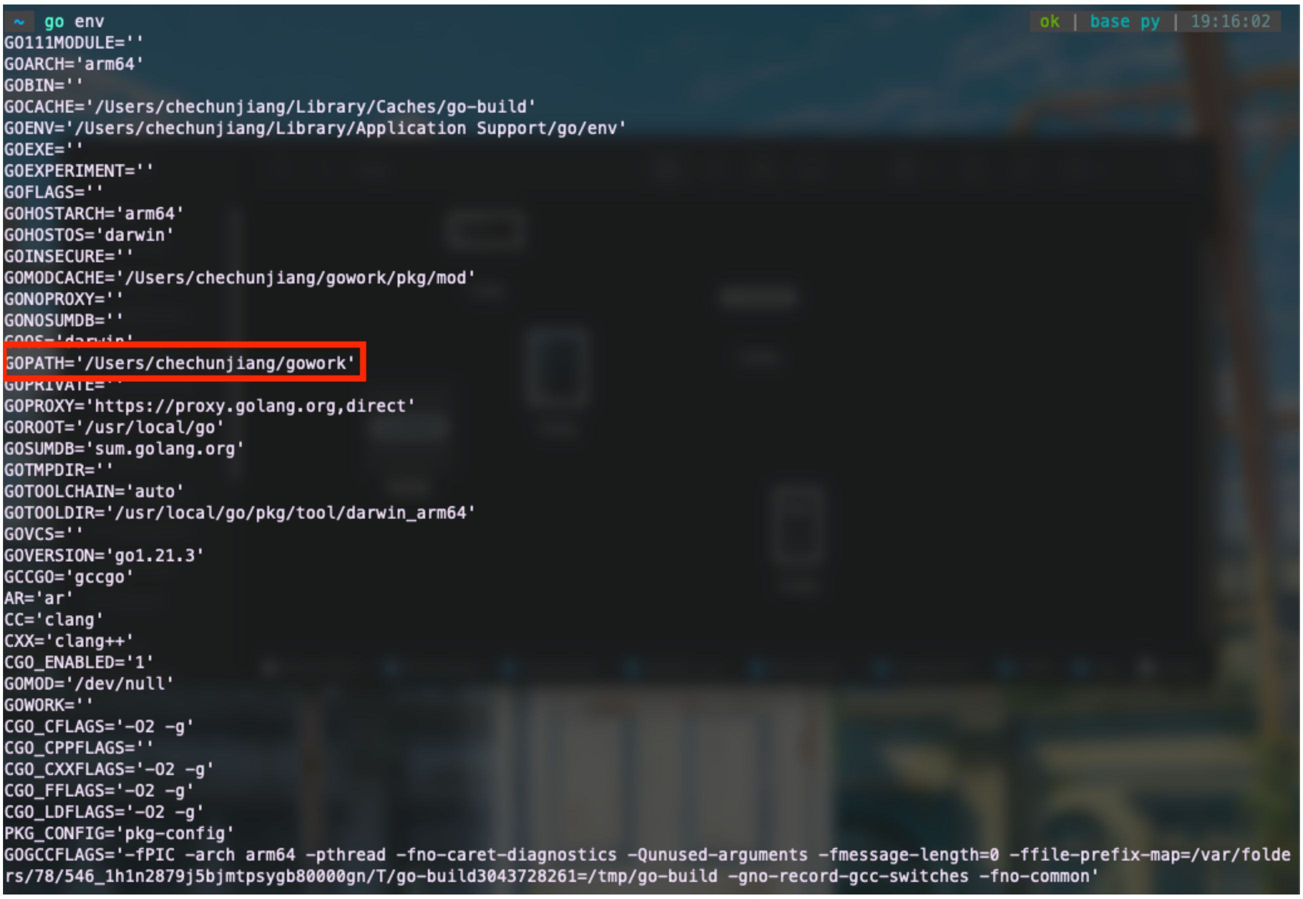
```
~ go version
go version go1.21.3 darwin/arm64
```

The Hello World Program

In this section, you get started with the Hello World program. Initially, you can set up the workspace of the Golang program by running the following commands.

```
mkdir $HOME/gowork
export GOPATH=$HOME/gowork
export PATH=$PATH:$GOPATH/bin
```

Then you could run `go env` to check whether the workspace is correctly set up.



```
~ go env                                                                                  ok | base py | 19:16:02
GO111MODULE=''
GOARCH='arm64'
GOBIN=''
GOCACHE='/Users/chechunjiang/Library/Caches/go-build'
GOENV='/Users/chechunjiang/Library/Application Support/go/env'
GOEXE=''
GOEXPERIMENT=''
GOFLAGS=''
GOHOSTARCH='arm64'
GOHOSTOS='darwin'
GOINSECURE=''
GOMODCACHE='/Users/chechunjiang/gowork/pkg/mod'
GONOPROXY=''
GONOSUMDB=''
GOOS='darwin'
GOPATH='/Users/chechunjiang/gowork'
GOPRIVATE=''
GOPROXY='https://proxy.golang.org,direct'
GOROOT='/usr/local/go'
GOSUMDB='sum.golang.org'
GOTMPDIR=''
GOTOOLCHAIN='auto'
GOTOOLDIR='/usr/local/go/pkg/tool/darwin_arm64'
GOVCS=''
GOVERSION='go1.21.3'
GCCGO='gccgo'
AR='ar'
CC='clang'
CXX='clang++'
CGO_ENABLED='1'
GOMOD='/dev/null'
GOWORK=''
CGO_CFLAGS='-O2 -g'
CGO_CPPFLAGS=''
CGO_CXXFLAGS='-O2 -g'
CGO_FFLAGS='-O2 -g'
CGO_LDFLAGS='-O2 -g'
PKG_CONFIG='pkg-config'
GOGCCFLAGS='-fPIC -arch arm64 -pthread -fno-caret-diagnostics -Qunused-arguments -fmessage-length=0 -ffile-prefix-map=/var/folde
rs/78/546_1h1n2879j5bjmtpsygb80000gn/T/go-build3043728261=/tmp/go-build -gno-record-gcc-switches -fno-common'
```

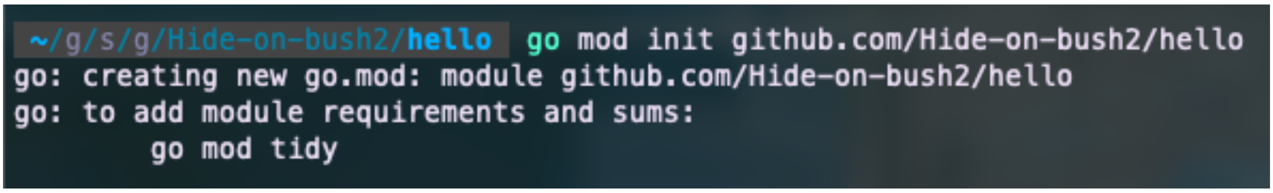
Subsequently, we create a directory for our Hello World program by following commands:

```
mkdir -p $GOPATH/src/github.com/Hide-on-bush2/hello
```

Golang relies on `mod` to manage the dependencies among different files. Specifically, when your code imports packages contained in other modules, you manage those dependencies through your code's own module. That module is defined by a `go.mod` file that tracks the modules that provide those packages. That `go.mod` file stays with your code, including in your source code repository. To enable dependency tracking for your code by creating a `go.mod` file, run the `go mod init` command, giving it the name of the module your code will be in. The name is the module's module path. For instance, we run

```
go mod init github.com/Hide-on-bush2/hello
```

The expected response is as follows:

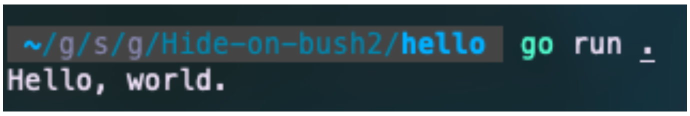


```
~/g/s/g/Hide-on-bush2/hello  go mod init github.com/Hide-on-bush2/hello
go: creating new go.mod: module github.com/Hide-on-bush2/hello
go: to add module requirements and sums:
        go mod tidy
```

Then you should go into the directory `github.com/Hide-on-bush2/hello` and create a file `hello.go` with the following content:

```
package main

import "fmt"

func main() {
  fmt.Printf("Hello, world.\\n")
}
```

Congratulations! Now you are able to run your first program by running `go run .` , you are expected to get the output as follows.



```
~/g/s/g/Hide-on-bush2/hello  go run .
Hello, world.
```

There are other options to run a Go program. Specifically, if you want to run Hello World in other directories, you are advised to run `go install github.com/Hide-on-bush2/hello` , which enables you to run Hello World by running `hello` in whatever directory you are located at.

Create a Library

Putting all codes into one file makes a program hard to manage and test. A better strategy is that codes of different modules are distributed into different files. A file could provide implemented functionalities for other files, referred to as a library. In this section, we provide the basic guidance for you to create the first library.

Firstly, we create a directory for our library.

```
mkdir $GOPATH/src/github.com/Hide-on-bush2/stringutil
```

Within this directory, we create a file `reverse.go` containing the following content:

```
package stringutil

func Reverse(s string) string {
  r := []rune(s)
  for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
    r[i], r[j] = r[j], r[i]
  }
  return string(r)
}
```

Similarly, we init the `mod` for this package

```
go mod init github.com/Hide-on-bush2/stringutil
```

Then we run the following command to compile the package:

```
go install github.com/Hide-on-bush2/stringutil
```

We now successfully create our first library, and then we try to import it into our Hello World program. Modify the `hello.go` as follows:

```
package main
import (
  "fmt"
  "github.com/Hide-on-bush2/stringutil"
)

func main() {
  fmt.Printf(stringutil.Reverse("!oG ,olleH"))
}
```

We import `stringutil` by the code of line 4. However, in the latest version of Golang, the compiler will find the declaimed package, `github.com/Hide-on-bush2/stringutil`, from the Github depository. The target package still remains locally. To import packages locally, we can modify the path to find the package manually. For instance, we run the following command to replace the path with `../stringutil`, which is a local directory.

```
go mod edit -replace github.com/Hide-on-bush2/stringutil=../s
```

Run `go install github.com/Hide-on-bush2/hello` to compile the program and `hello` to run it.

Test

In this section, you are guided to write an unit test for the `stringutil` package. Specifically, create a file `reverse_test.go` in `stringutil` directory, containing the following content:

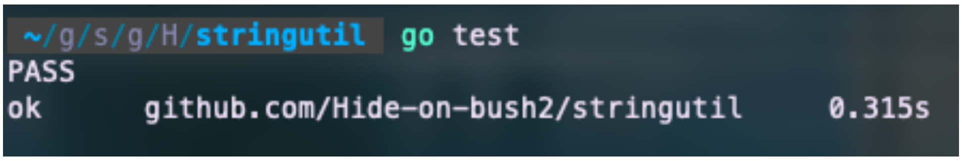
```
package stringutil

import "testing"

func TestReverse(t *testing.T) {
  cases := []struct {
    in, want string
  }{
    {"Hello, world", "dlrow ,olleH"},
    {"Hello, 世界", "界世 ,olleH"},
    {"", ""},
  }
  for _, c := range cases {
    got := Reverse(c.in)
    if got != c.want {
      t.Errorf("Reverse(%q) == %q, want %q", c.in, got, c.wan
        }
      }
    }
```

Now you are able to check whether your implementation passes the designed unit test by runging

```
go test github.com/Hide-on-bush2/stringutil
```

If you are locating at the `stringutil` directory, you can achieve the same goal by running `go test`

The expected output is as follows:



```
~/g/s/g/H/stringutil  go test
PASS
ok      github.com/Hide-on-bush2/stringutil     0.315s
```