**Go directives**

**Retract directive**

It means draw back or with draw

Let us assume we publish our module using version control mechanism

In one module suppose did a mistake and released to production with number v0.1.0

After that realise a mistake and publish a new version with v0.2.0

We cant modify the cose in v0.1.0

And there is no way to tell the people that use v0.2.0

This problem will solved by the retract module

Can upgrade module

Can downgrade  modules

**Go module directives**

Applicable in and after version 1.13 of go

It is the new way of adding libraries called go modules

Go module solves the gopath problems

package main

import (

    "fmt"

    "mymodule/mypackage"

)

func main() {

    fmt.Println("Hello, Modules!")

    mypackage.PrintHello()

}

package mypackage

import "fmt"

func PrintHello() {

    fmt.Println("Hello, Modules! This is mypackage speaking!")

}

    Directory: C:\Go\_WorkSpace\projects\mymodules

Mode                 LastWriteTime         Length Name

----                 -------------         ------ ----

d-----        10-05-2022  02:18 PM                mypackage

-a----        10-05-2022  02:14 PM             25 go.mod

-a----        10-05-2022  02:18 PM            142 main.go

 Taken reference from <https://www.digitalocean.com/community/tutorials/how-to-use-go-modules>

**Adding a remote module as a dependencies**

go get github.com/spf13/cobra@07445ea

module mymodule

go 1.16

require (

github.com/inconshreveable/mousetrap v1.0.0 // indirect

github.com/spf13/cobra v1.1.2-0.20210209210842-07445ea179fc // indirect

github.com/spf13/pflag v1.0.5 // indirect

)

go get github.com/spf13/cobra@v1.1.1

module mymodule

go 1.16

require (

github.com/inconshreveable/mousetrap v1.0.0 // indirect

github.com/spf13/cobra v1.1.1 // indirect

github.com/spf13/pflag v1.0.5 // indirect

)

go get github.com/spf13/cobra@latest

module mymodule

go 1.16

require (

github.com/inconshreveable/mousetrap v1.0.0 // indirect

github.com/spf13/cobra v1.2.1 // indirect

github.com/spf13/pflag v1.0.5 // indirect

)

**Replace directory**

Replace directory will replace the content of the specific version of the midule from other wheres.

If the version present on the left side of the arrow only that specific version is replaced

Replace directory only applied on the main modules go.mod file, ignored by others

If there is multiple main than it will appy to all

right habd side begin with ./ or ../ then it is local path for replacement

Example:

replace golang.org/x/net v1.2.3 => example.com/fork/net v1.4.5

replace (

    golang.org/x/net v1.2.3 => example.com/fork/net v1.4.5

    golang.org/x/net => example.com/fork/net v1.4.5  //Module with no local path

    golang.org/x/net v1.2.3 => ./fork/net

    golang.org/x/net => ./fork/net   // Local path

)

**Exclude directory**

Exclude directory prevents the module version loaded by go command

Before Go 1.16 exclude version was reference by the required directory

Exclude directory only applied in main go.mod it will be ignored by others

ExcludeDirective = "exclude" ( ExcludeSpec | "(" newline { ExcludeSpec } ")" newline ) .

ExcludeSpec = ModulePath Version newline .

Example:

exclude golang.org/x/net v1.2.3

exclude (

    golang.org/x/crypto v1.4.5

    golang.org/x/text v1.6.7

)

**Require directory**

Required directory declares the minimum required version of the given module dependencies.

Go command loads the go.mod file for required version & incorporate requirement from the file

Go command will automatically adds // indirect comments

Which indicates that no package from the required modules is directly imported by any package in main module

RequireDirective = "require" ( RequireSpec | "(" newline { RequireSpec } ")" newline ) .

RequireSpec = ModulePath Version newline .

Example:

require golang.org/x/net v1.2.3

require (

    golang.org/x/crypto v1.4.5 // indirect

    golang.org/x/text v1.6.7

)

**Go directives**

Go directive indicates that a module was written assuming the semantic of a given version of go.

The version is like 1.9, 1.14 etc

The go directive originally intended to support backward incompatibility changes to the go language.

There have no been incompatible language changes since modules was introduced, go directory still affects new language supports

The go.mod file after 1.17 includes an explicite require directive for each module that provides any package transively import by package or test in main module.

As of the Go 1.17 release, if the go directive is missing, go 1.16 is assumed.

GoDirective = "go" GoVersion newline .

GoVersion = string | ident .  /\* valid release version; see above \*/

Example:

go 1.14