GO

WORKSPACE

HOW TO WRITE GO CODE

- https://go.dev/doc/code

REPOSITORIO:
UNO O MÁS MÓDULOS

MODULO:
COLECCIÓN DE PAQUETES

PAQUETE:
COLECCIÓN DE ARCHIVOS

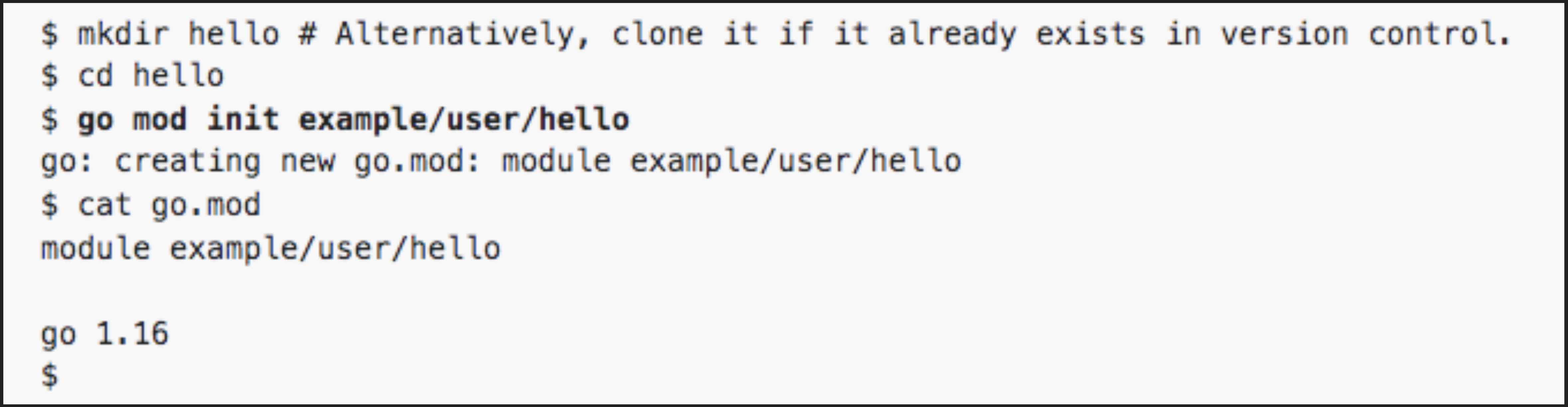
HOW TO WRITE GO CODE

- Archivo go.mod

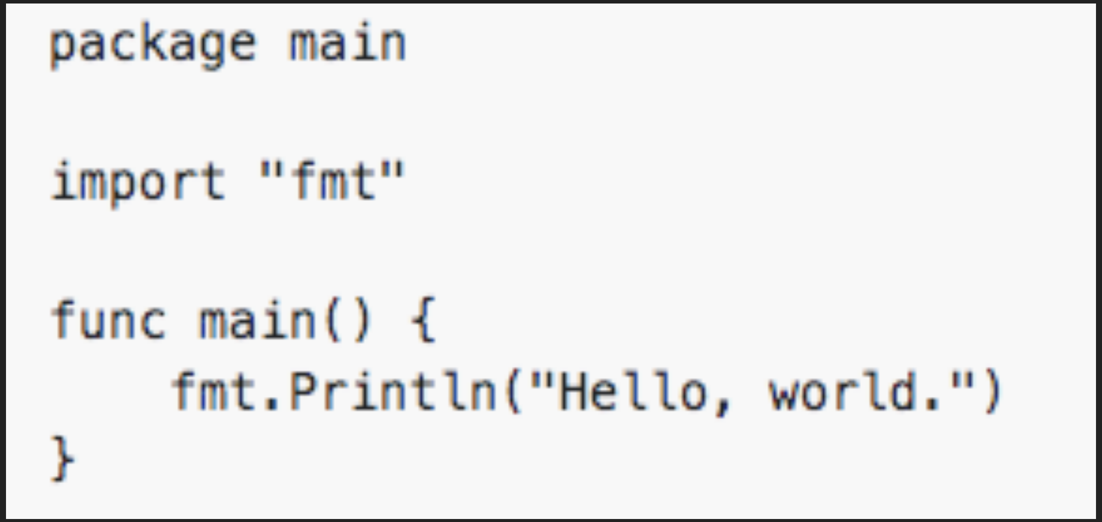
  Declara la ruta del módulo.

  No se necesita publicar el código en un repositorio remoto antes de poder compilarlo. Un módulo se puede definir localmente sin pertenecer a un repositorio.

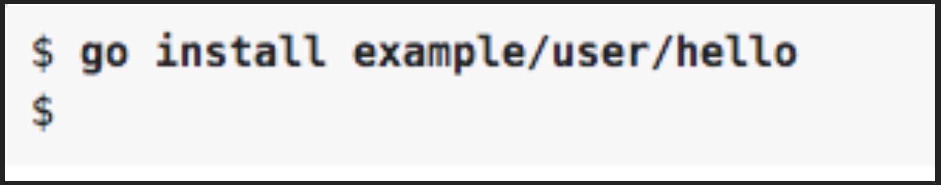
HOW TO WRITE GO CODE



```
$ mkdir hello # Alternatively, clone it if it already exists in version control.
$ cd hello
$ go mod init example/user/hello
go: creating new go.mod: module example/user/hello
$ cat go.mod
module example/user/hello

go 1.16
$
```

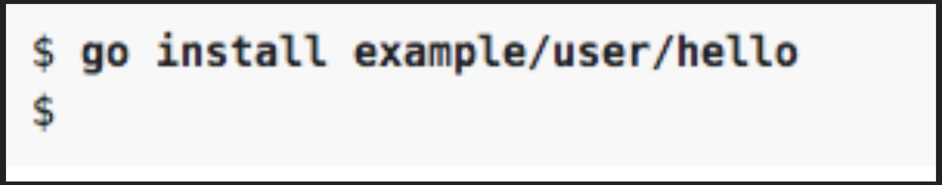


```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, world.")
}
```



```
$ go install example/user/hello
$
```

HOW TO WRITE GO CODE



```
$ go install example/user/hello
$
```

- This command builds the `hello` command, producing an executable binary. It then installs that binary as `$HOME/go/bin/hello`
- The install directory is controlled by the GOPATH andGOBIN environment variables.
- If GOBIN is set, binaries are installed to that directory.
- If GOPATH is set, binaries are installed to the bin subdirectory of the first directory in the GOPATH list. Otherwise, binaries are installed to the bin subdirectory of the default GOPATH ($HOME/GO)

GO WORKSPACE

- Folders are packages
- Every file in a folder/package must have the same package name
- Func main:
    - package.function
    - Capitalization: Visible/notVisible outside package
-

BASH

- go mod init _______(name)

GO MOD

go mod <command> [arguments]

Commands:

download	download modules to local cache
edit	edit go.mod from tools or scripts
graph	print module requerimiento graph
init	initialize new module in current directory
tidy	add missing and remove unused modules
vender	make vendored copy of dependencies
verify	verify dependencies have expected content
why	explain why packages or modules are needed

GO MOD

> go mod tidy:

It makes sure go.mod matches the source code in the module and removes requirements on modules that aren't used anymore

(go.sum ! Git and go dependencies)

GO CLEAN

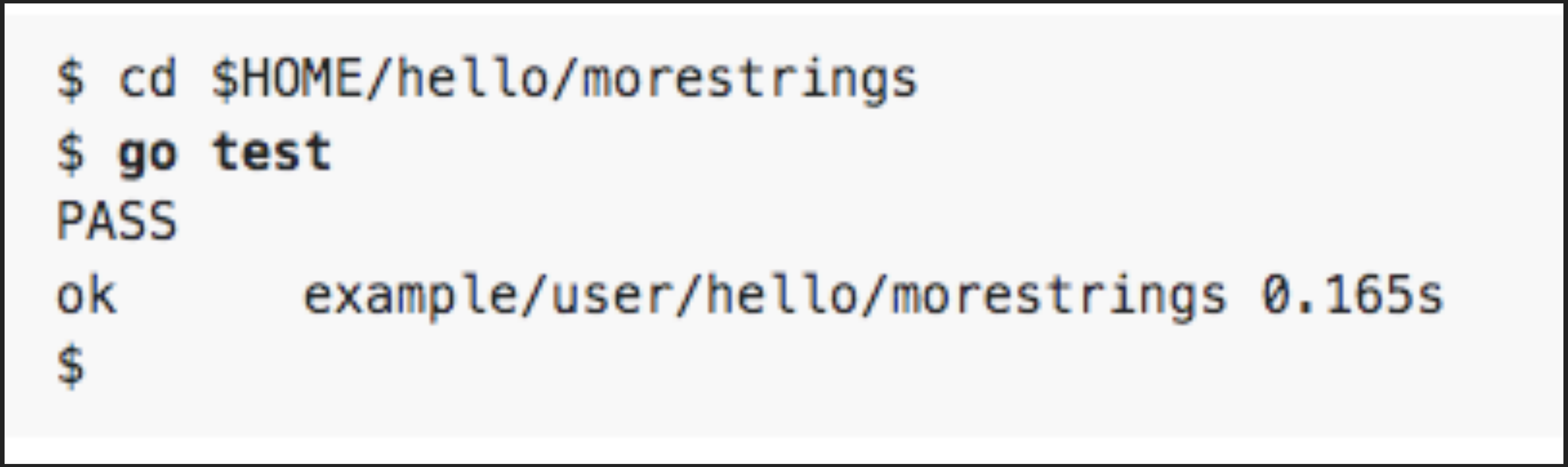
```
> go clean -modcache
```

To remove all downloaded modules,

GO TESTING

Go has a lightweight test framework composed of the `go test` command and the `testing` package.

- Name: name_test.go
- func: func (t *testing.T)

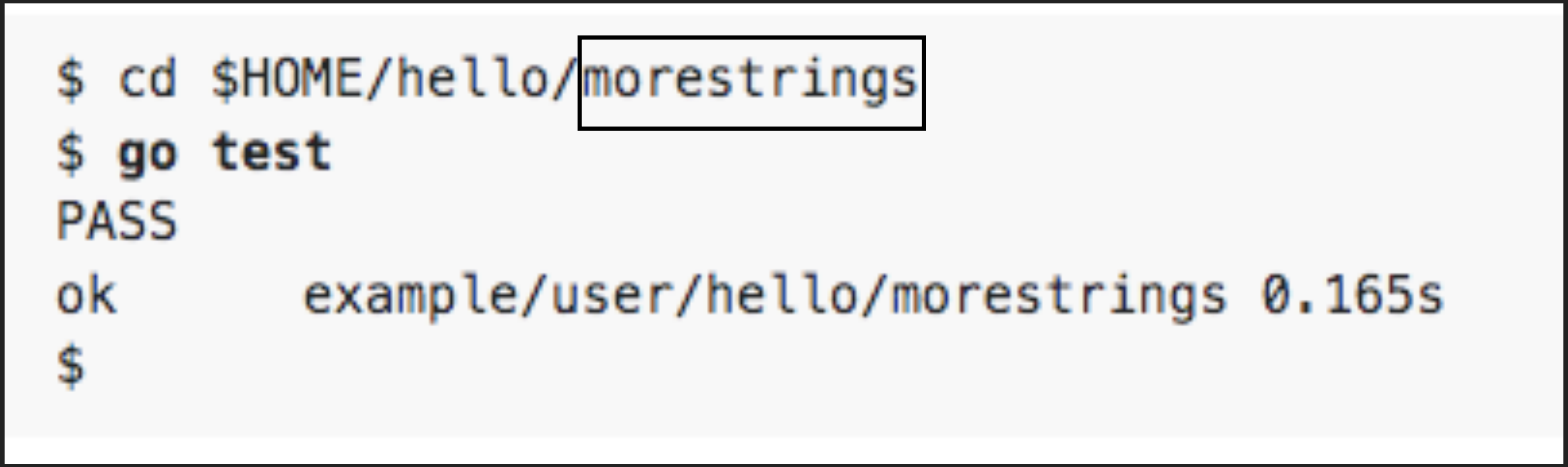


```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/morestrings 0.165s
$
```

GO TESTING

Go has a lightweight test framework composed of the `go test` command and the `testing` package.

- Name: name_test.go
- func: func (t *testing.T)

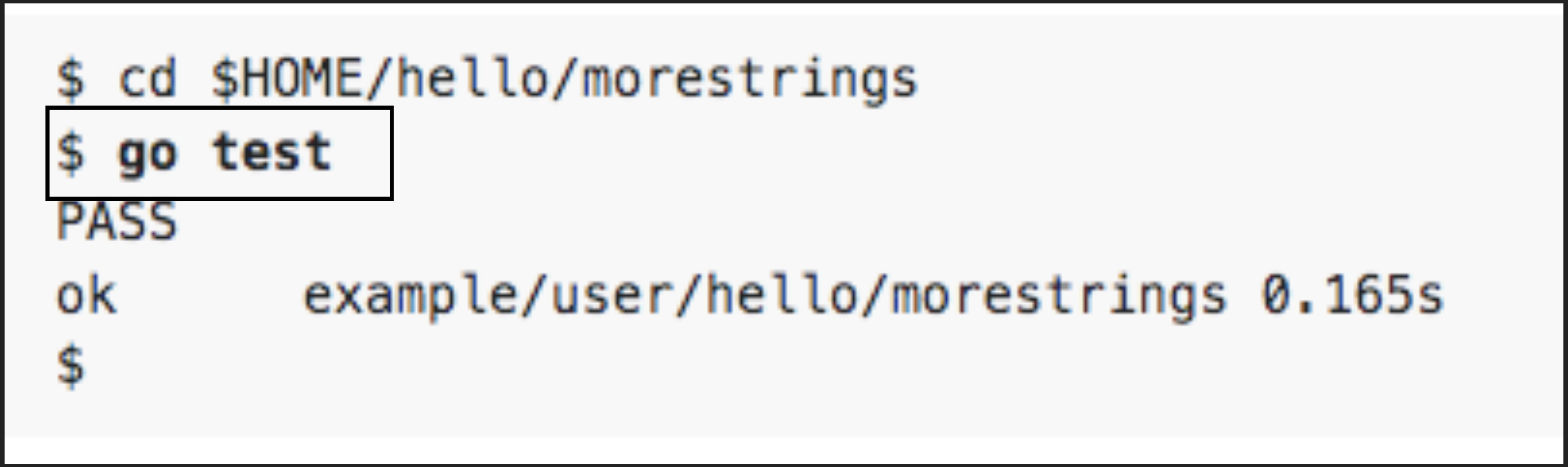


```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/morestrings 0.165s
$
```

GO TESTING

Go has a lightweight test framework composed of the `go test` command and the `testing` package.

- Name: name_test.go
- func: func (t *testing.T)

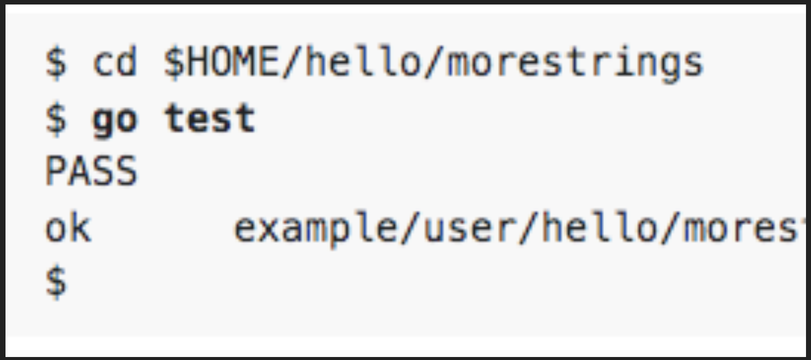


```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/morestrings 0.165s
$
```

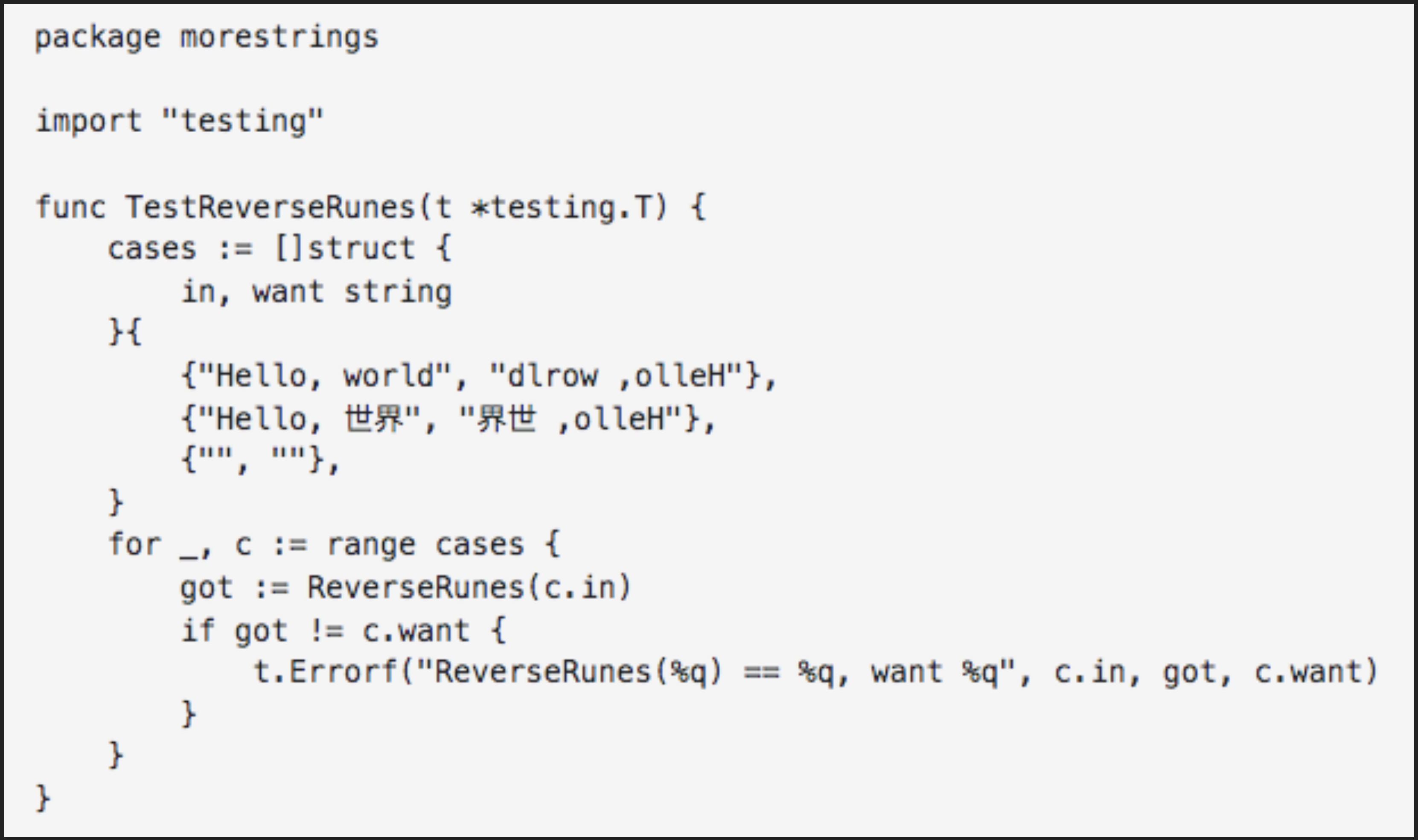
GO TESTING

Go has a lightweight test framework composed of the `go test` command and the `testing` package.

- Name: name_test.go
- func: func (t *testing.T)



```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/mores
$
```

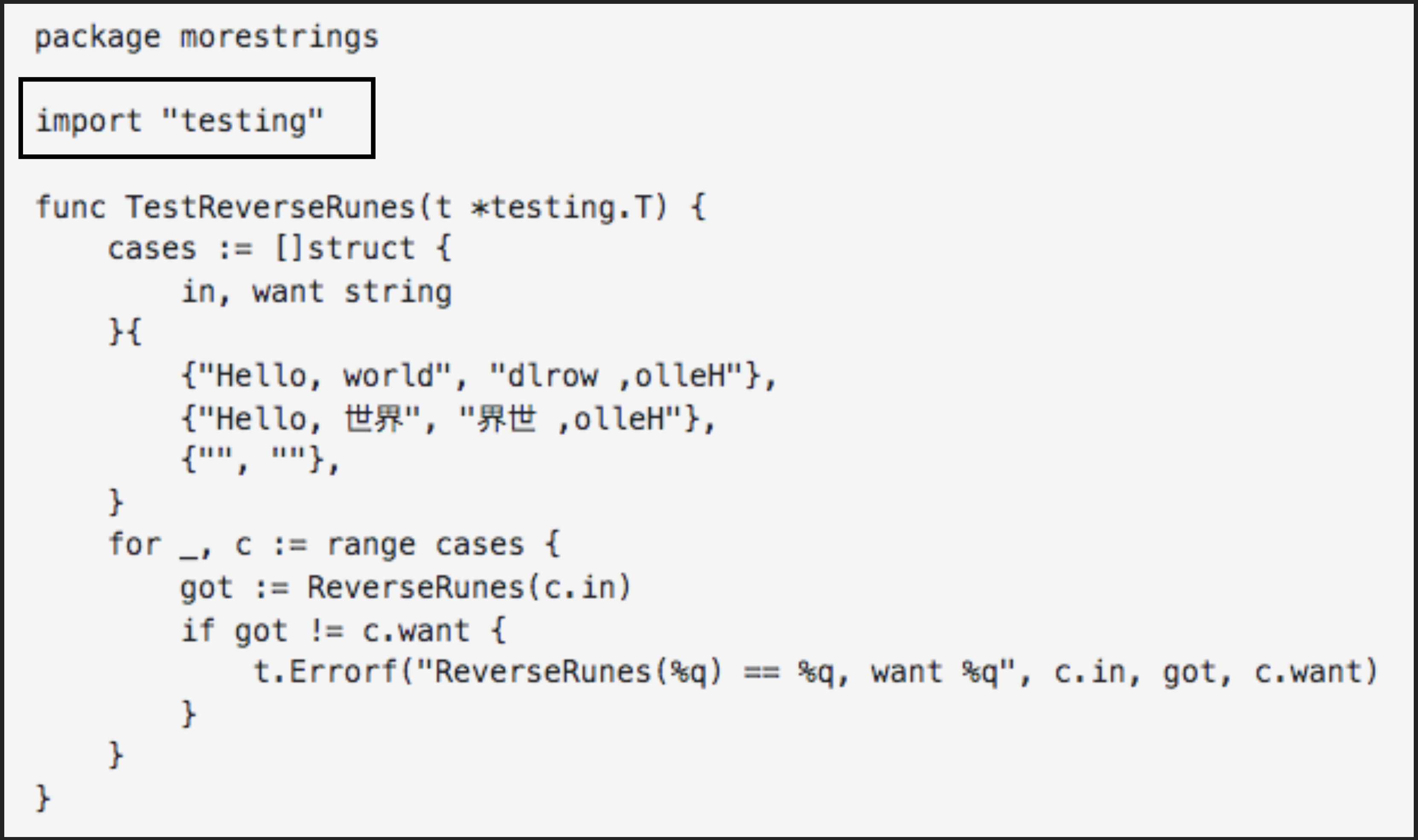


```
package morestrings

import "testing"

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

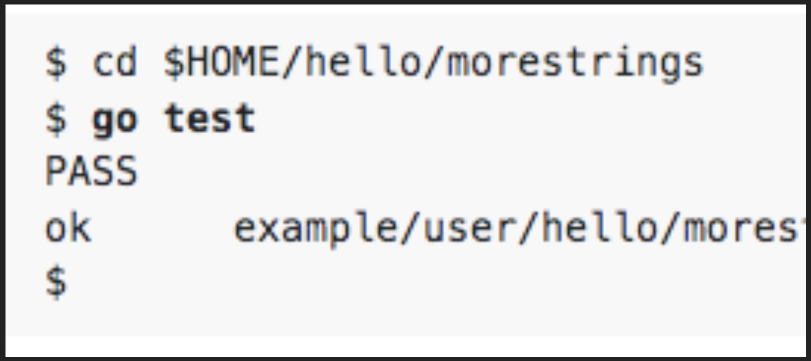
GO TESTING

Go has a lightweight test framework composed of the `go test` command and the `testing` package.

- Name: name_test.go
- func: func (t *testing.T)



```go
package morestrings

import "testing"

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

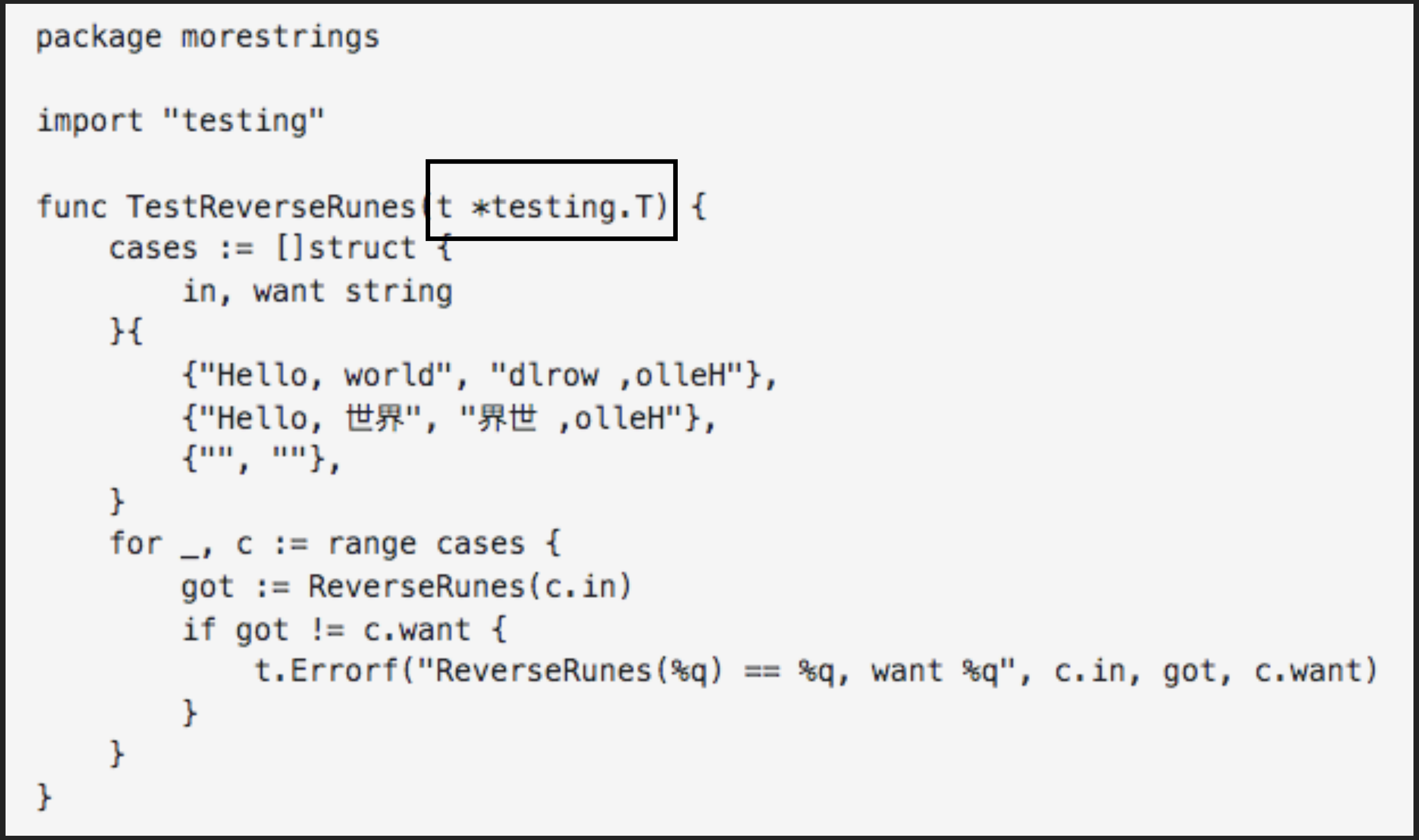


```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/mores
$
```

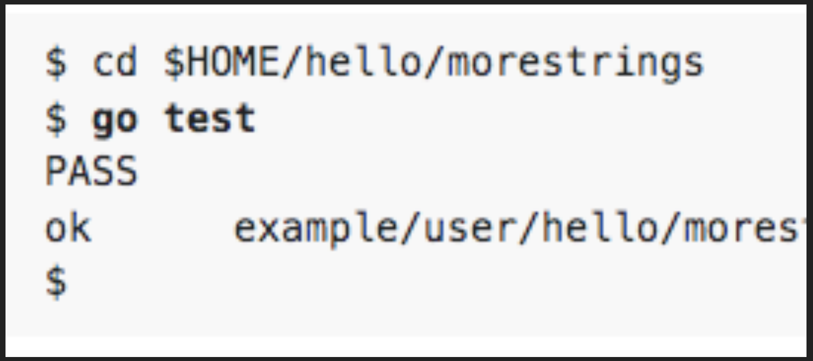
GO TESTING

Go has a lightweight test framework composed of the go test command and the testing package.

- Name: name_test.go
- func: func (t *testing.T)



```
package morestrings

import "testing"

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



```
$ cd $HOME/hello/morestrings
$ go test
PASS
ok      example/user/hello/mores
$
```

GO GET

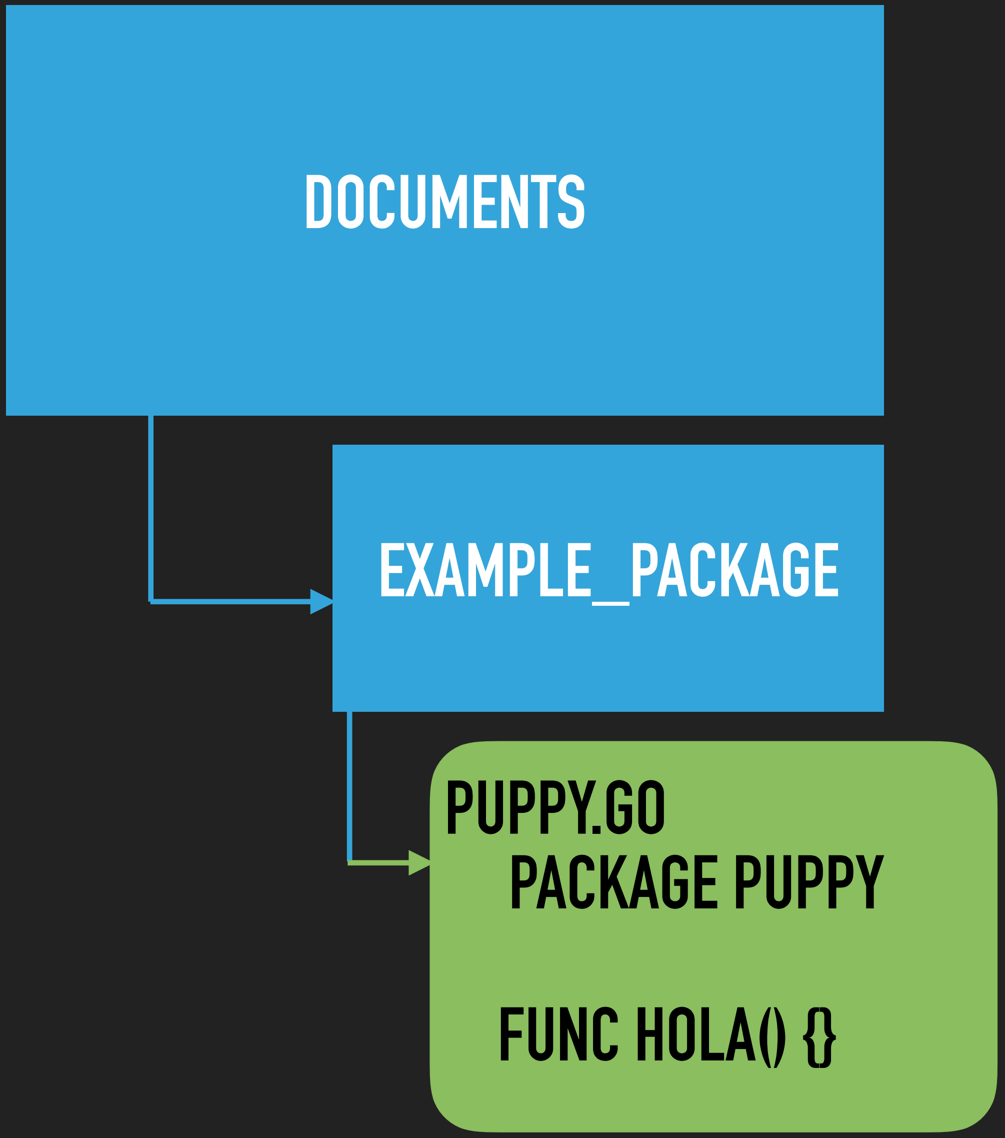
go get gitHub.com/name/folder

Get resolves its command-line arguments to packages at specific module versions, updates go.mod to require those versions, and downloads source code into the module cache.

It update the go.mod file.

- We can specify the latest version with
  - @latest ,
  - @commit_number

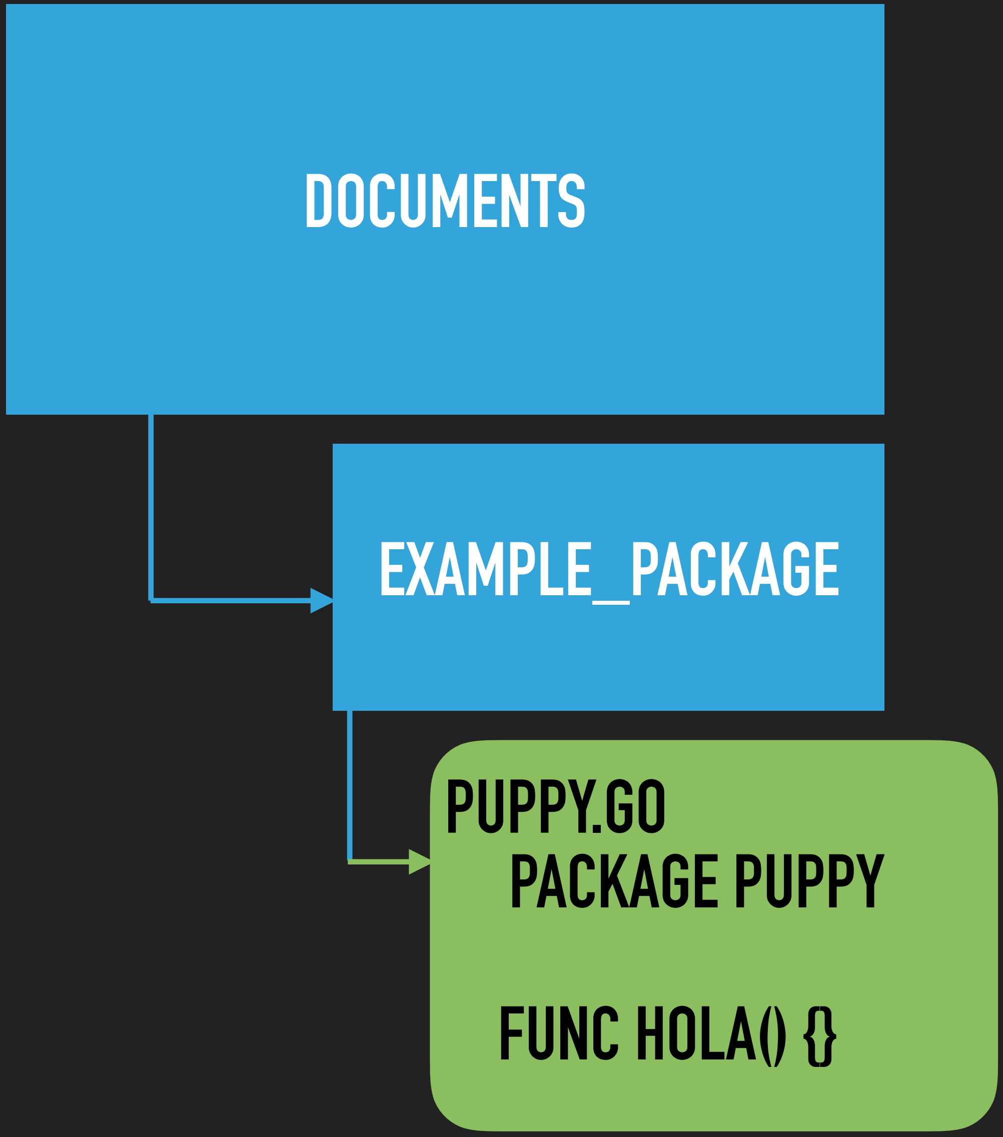
EJEMPLO: IMPORTING PACKAGES FROM REMOTE MODULES



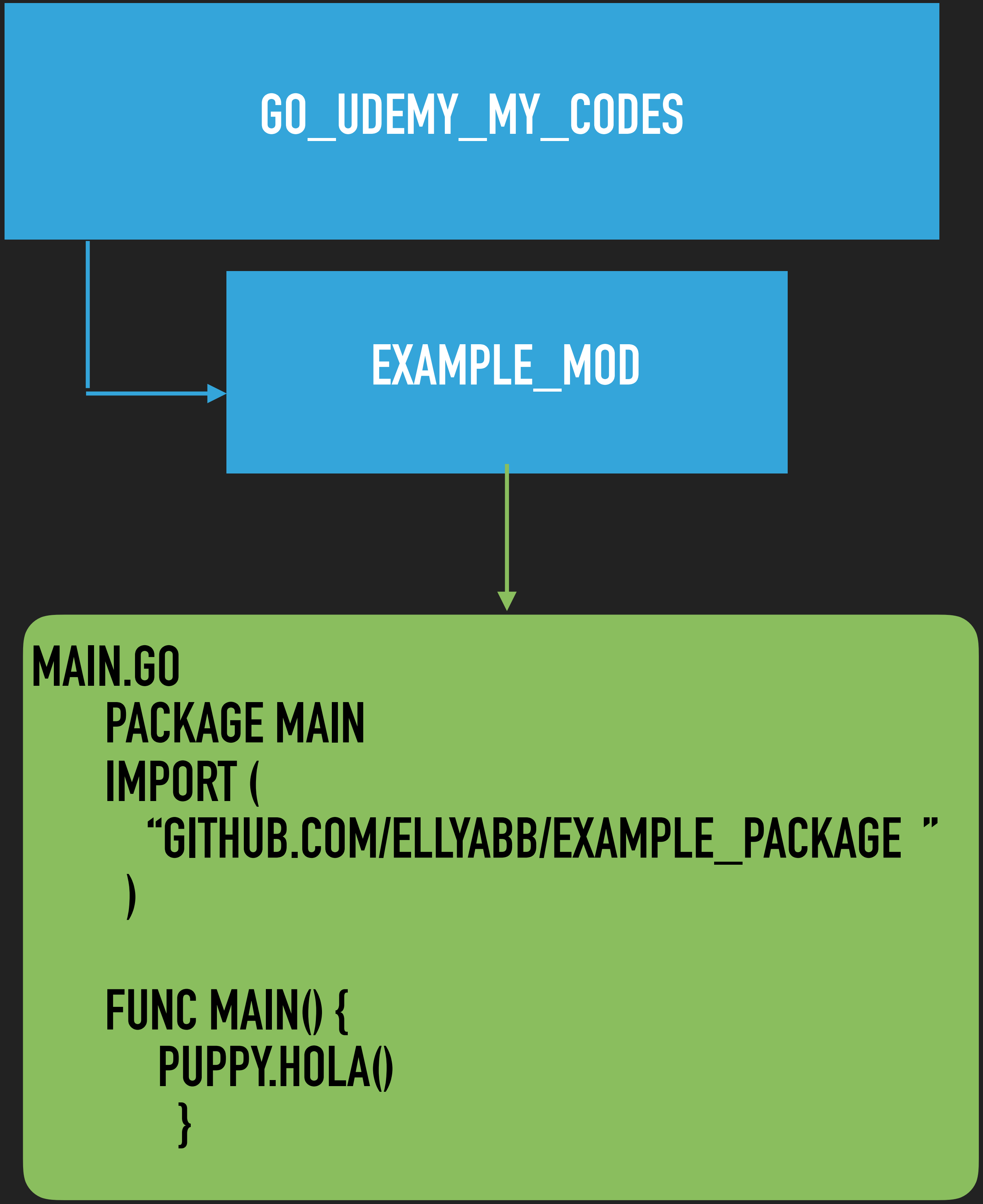
git init

gitHub.com/EllyABB/Example_package

EJEMPLO



```
PUPPY.GO
   PACKAGE PUPPY

  FUNC HOLA() {}
```

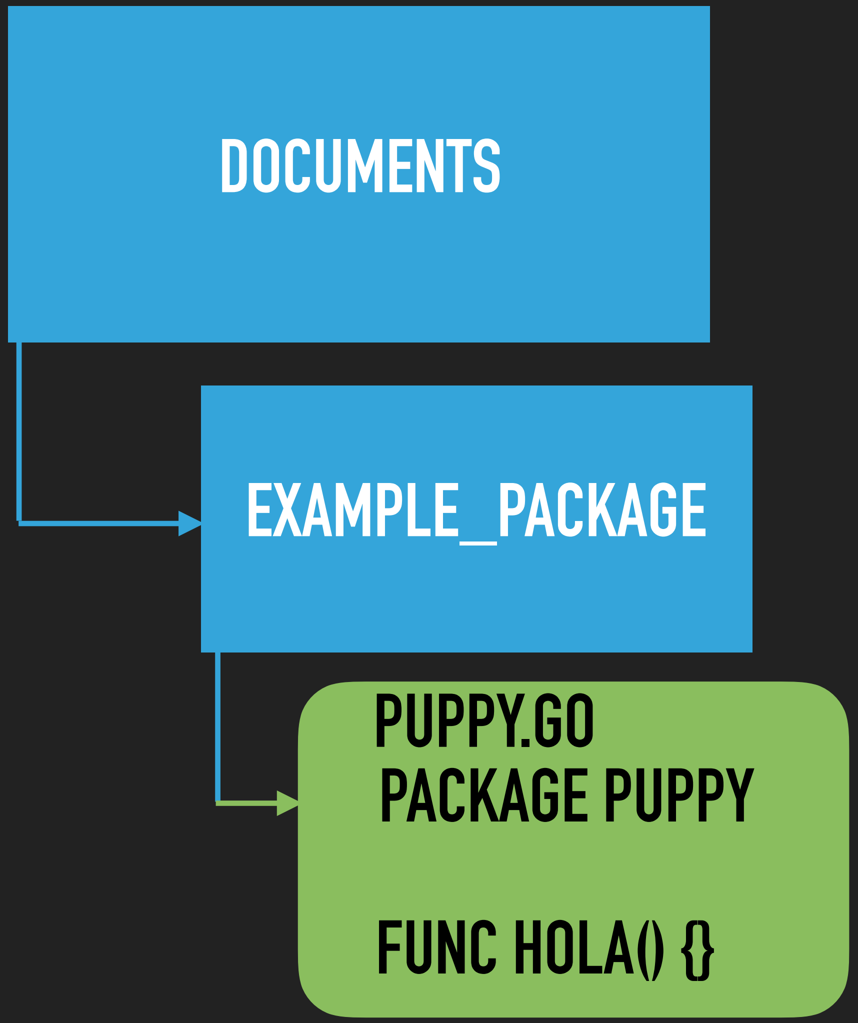


```
MAIN.GO
     PACKAGE MAIN
     IMPORT (
        "GITHUB.COM/ELLYABB/EXAMPLE_PACKAGE "
      )

     FUNC MAIN() {
         PUPPY.HOLA()
          }
```

git init

gitHub.com/EllyABB/Example_package

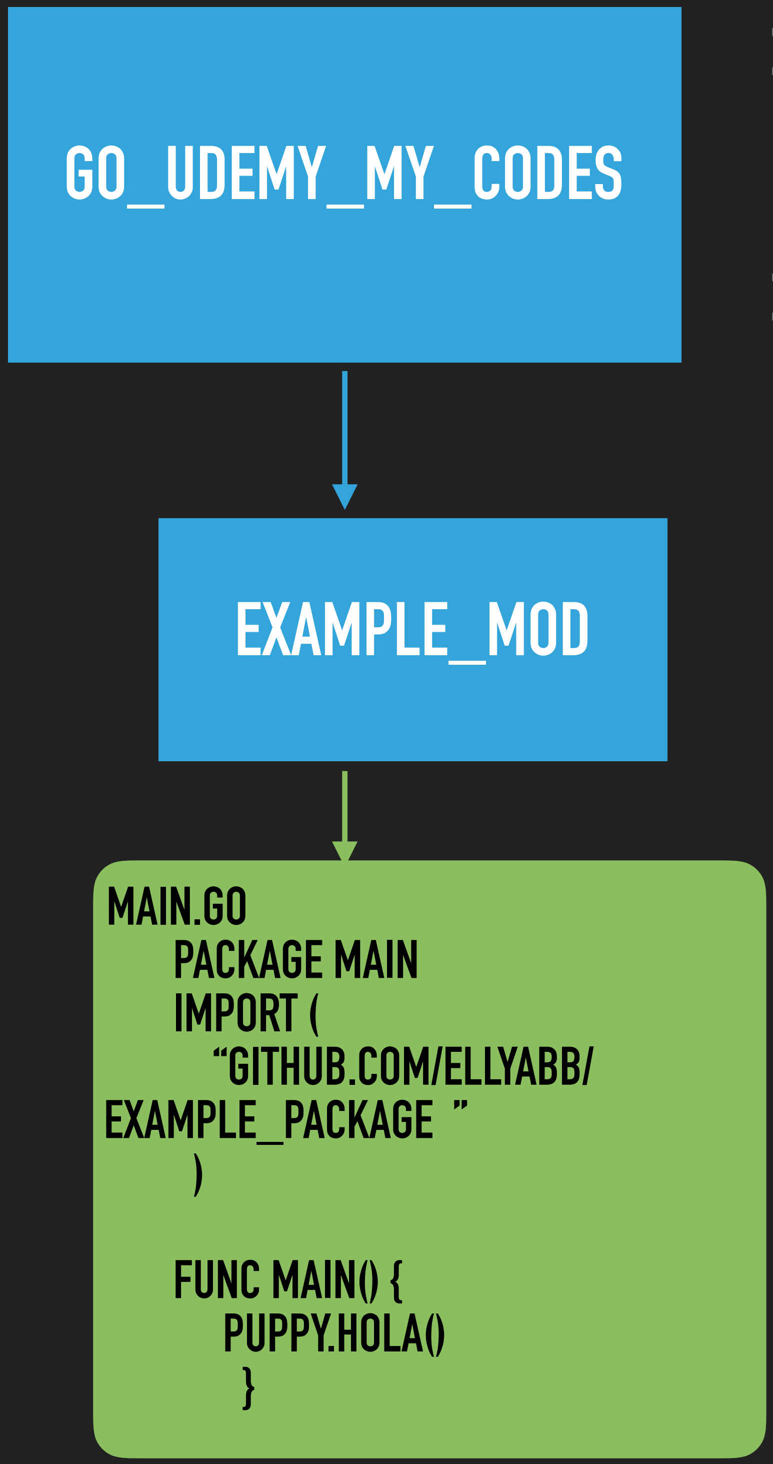
EJEMPLO



```
PUPPY.GO
PACKAGE PUPPY

FUNC HOLA() {}
```

>git init

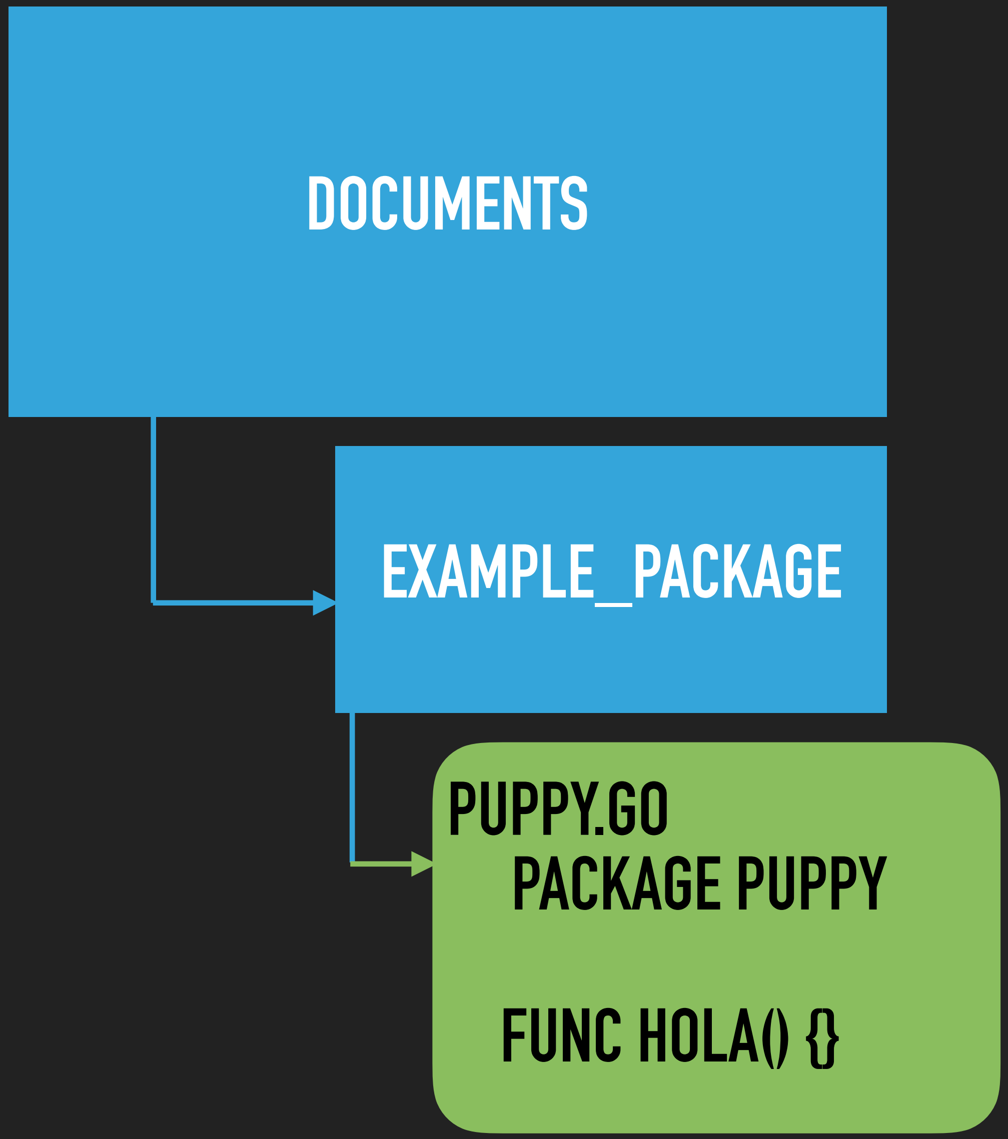
gitHub.com/EllyABB/Example_package



```
MAIN.GO
    PACKAGE MAIN
    IMPORT (
        "GITHUB.COM/ELLYABB/
EXAMPLE_PACKAGE "
      )

    FUNC MAIN() {
        PUPPY.HOLA()
        }
```

```
>go mod init mycodes

>go get GitHub.com/E..
/example
```

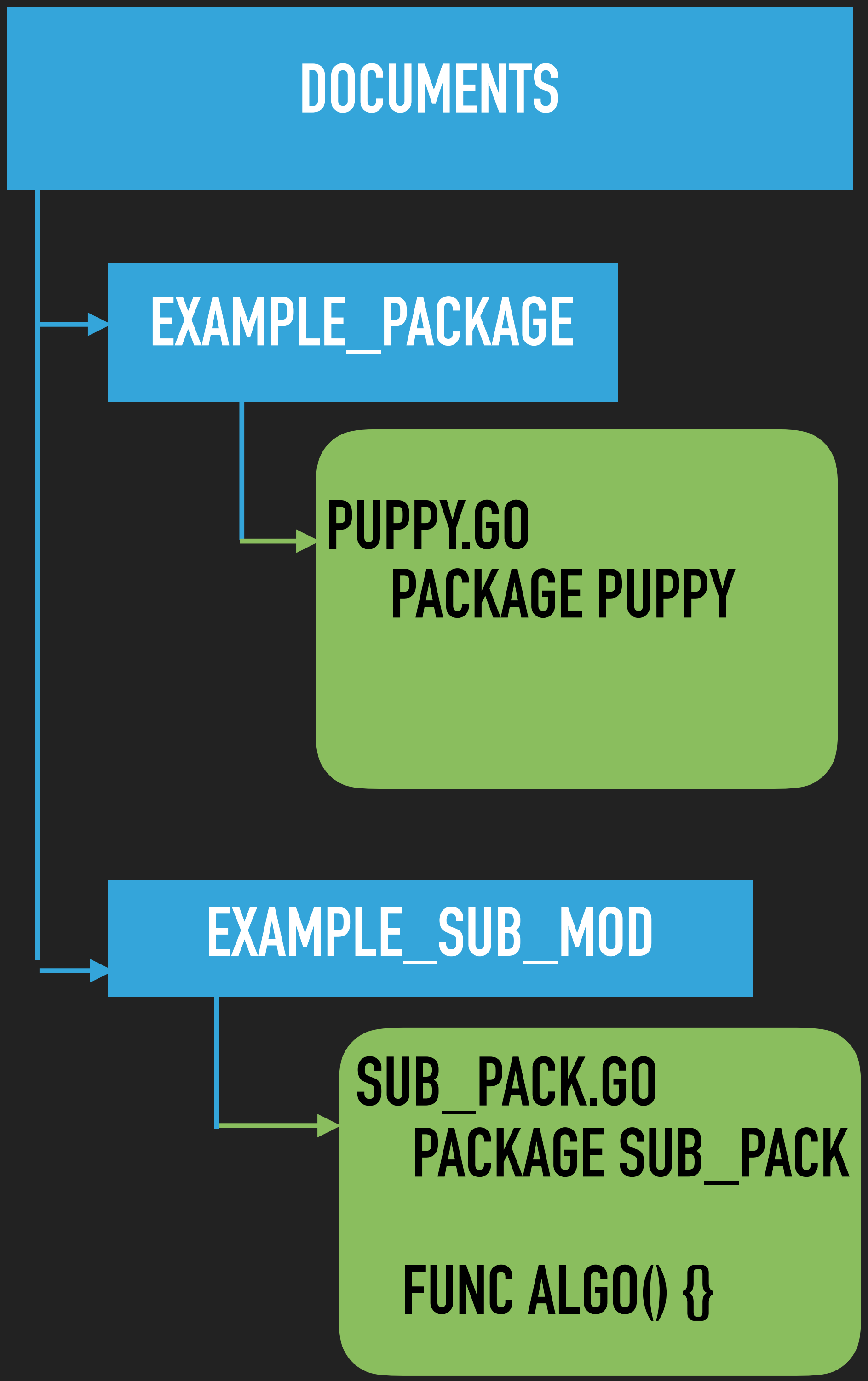
EJEMPLO



```
> git init gitHub.com/EllyABB/Example_package
```

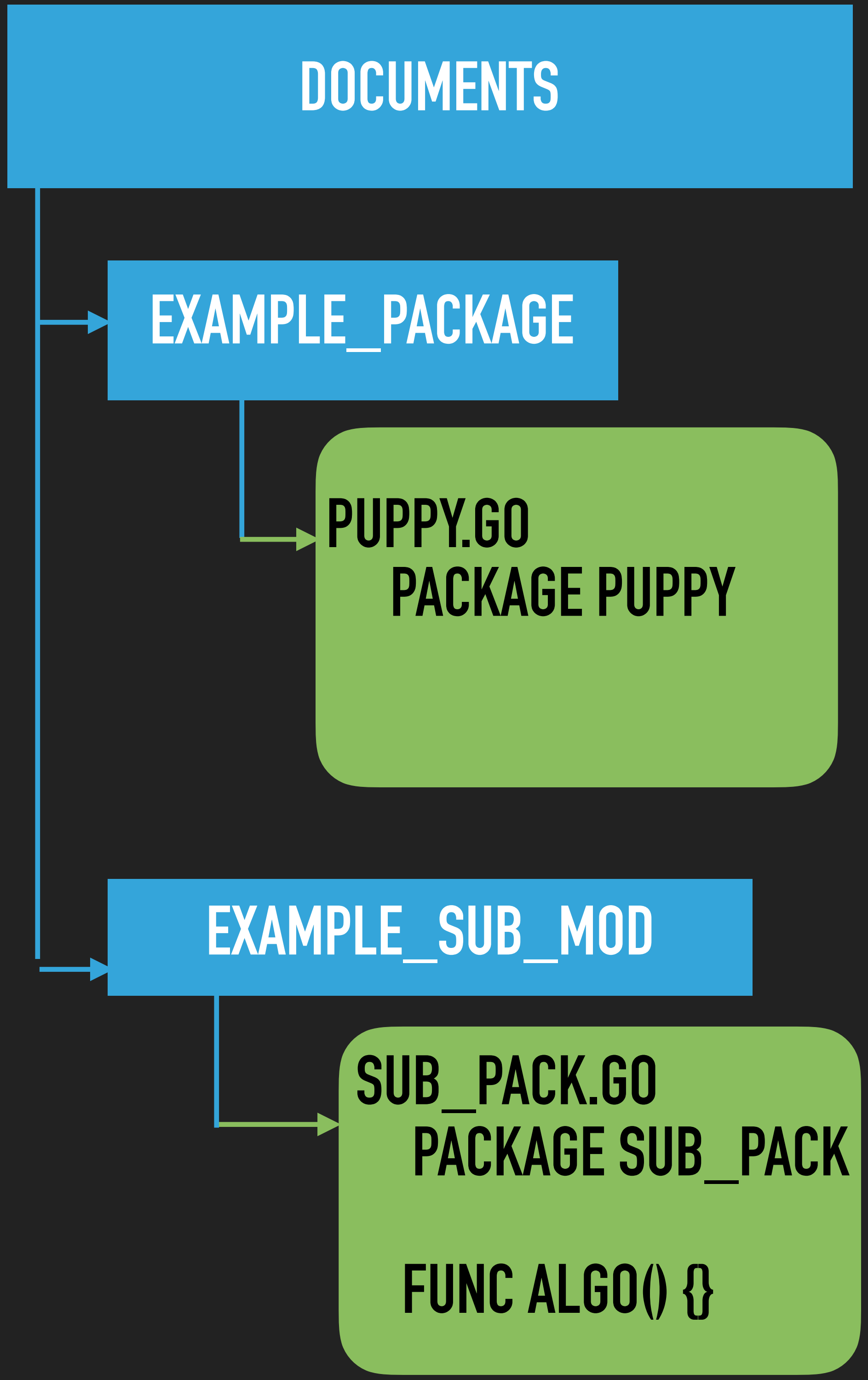
```
> go mod init gitHub.com/EllyABB/Example_package
```

EJEMPLO 2



```
> git init gitHub.com/EllyABB/Example_sub_package@#commit

> go mod init gitHub.com/EllyABB/Example_sub_package
```

EJEMPLO 2

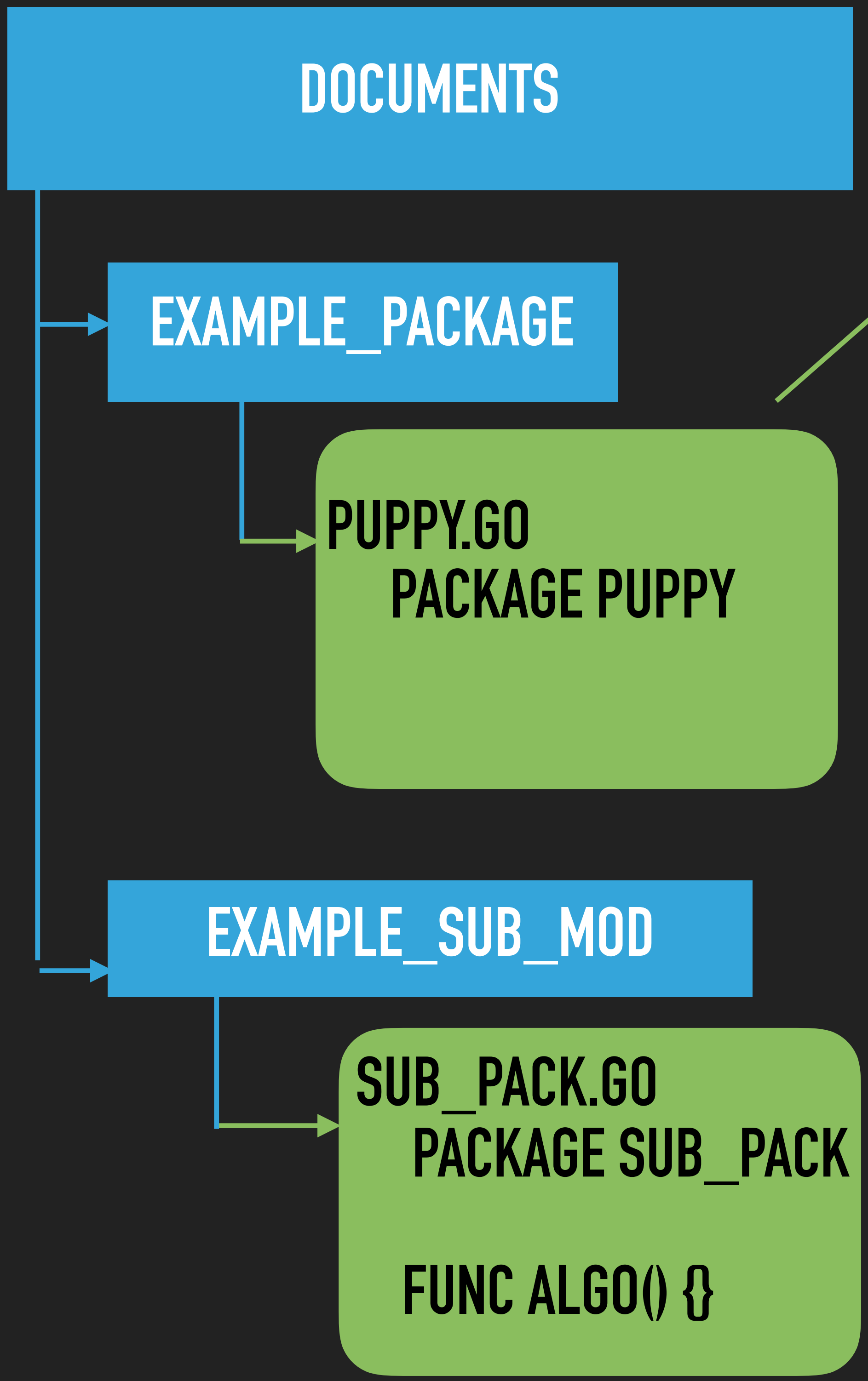


```
> go get gitHub.com/EllyABB/Example_sub_pack
@#commit
> git
> go mod tidy
```

```
> git init gitHub.com/EllyABB/
Example_sub_package@#commit
```

```
> go mod init gitHub.com/EllyABB/
Example_sub_package
```

EJEMPLO 2



```
PUPPY.GO
    PACKAGE PUPPY
```

```
SUB_PACK.GO
    PACKAGE SUB_PACK

    FUNC ALGO() {}
```

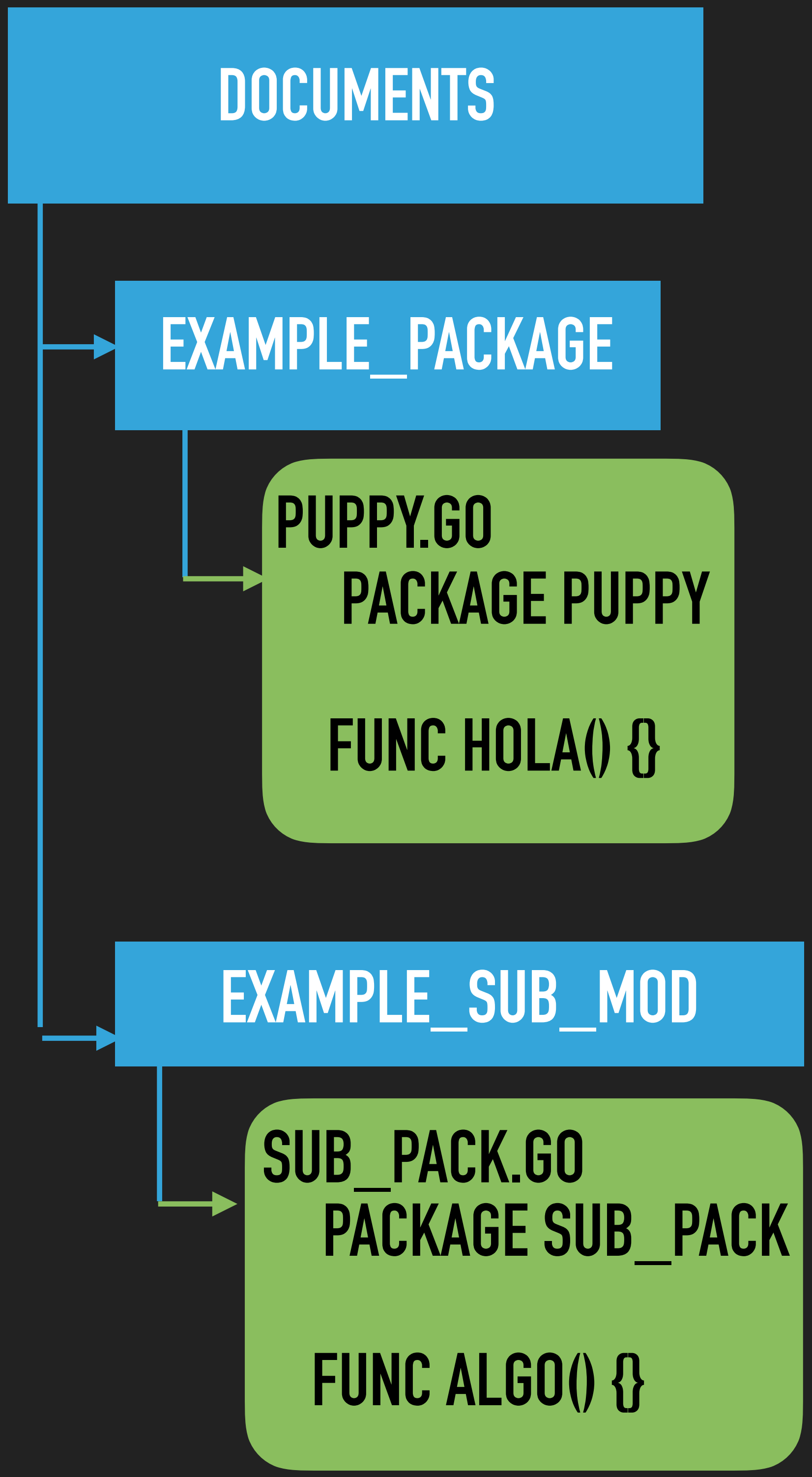
```
PUPPY.GO
    PACKAGE PUPPY
    IMPORT (
        "GITHUB.COM/ELLYABB/EXAMPLE_PACKAGE "
    )
   FUNC HOLA() {}
```

```
> go get gitHub.com/EllyABB/Example_sub_pack
@#commit
> git
> go mod tidy
```

```
> git init gitHub.com/EllyABB/
Example_sub_package@#commit
```

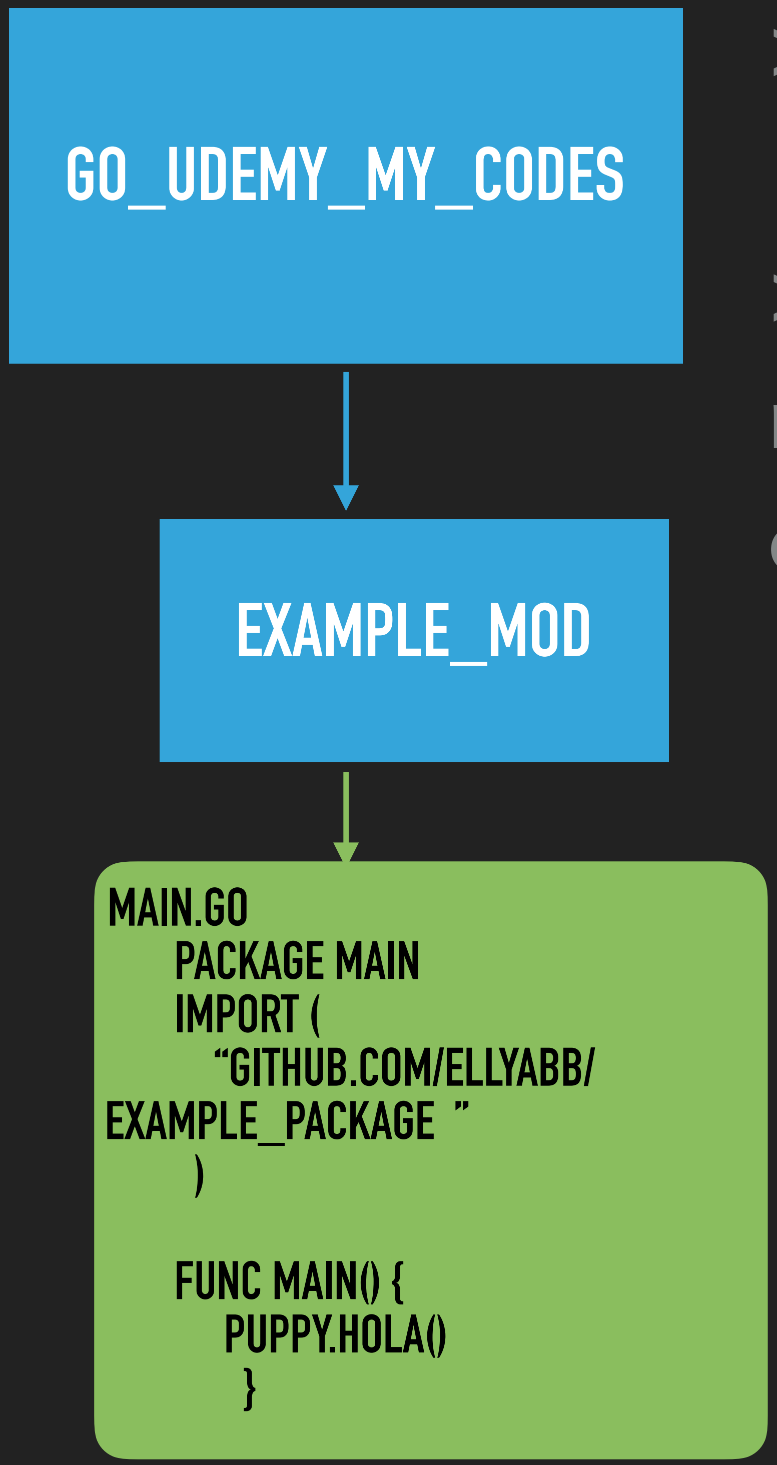
```
> go mod init gitHub.com/EllyABB/
Example_sub_package
```

EJEMPLO 2



```
PUPPY.GO
    PACKAGE PUPPY

    FUNC HOLA() {}
```

```
SUB_PACK.GO
    PACKAGE SUB_PACK

    FUNC ALGO() {}
```



```
MAIN.GO
    PACKAGE MAIN
    IMPORT (
        “GITHUB.COM/ELLYABB/
EXAMPLE_PACKAGE ”
    )

    FUNC MAIN() {
        PUPPY.HOLA()
    }
```

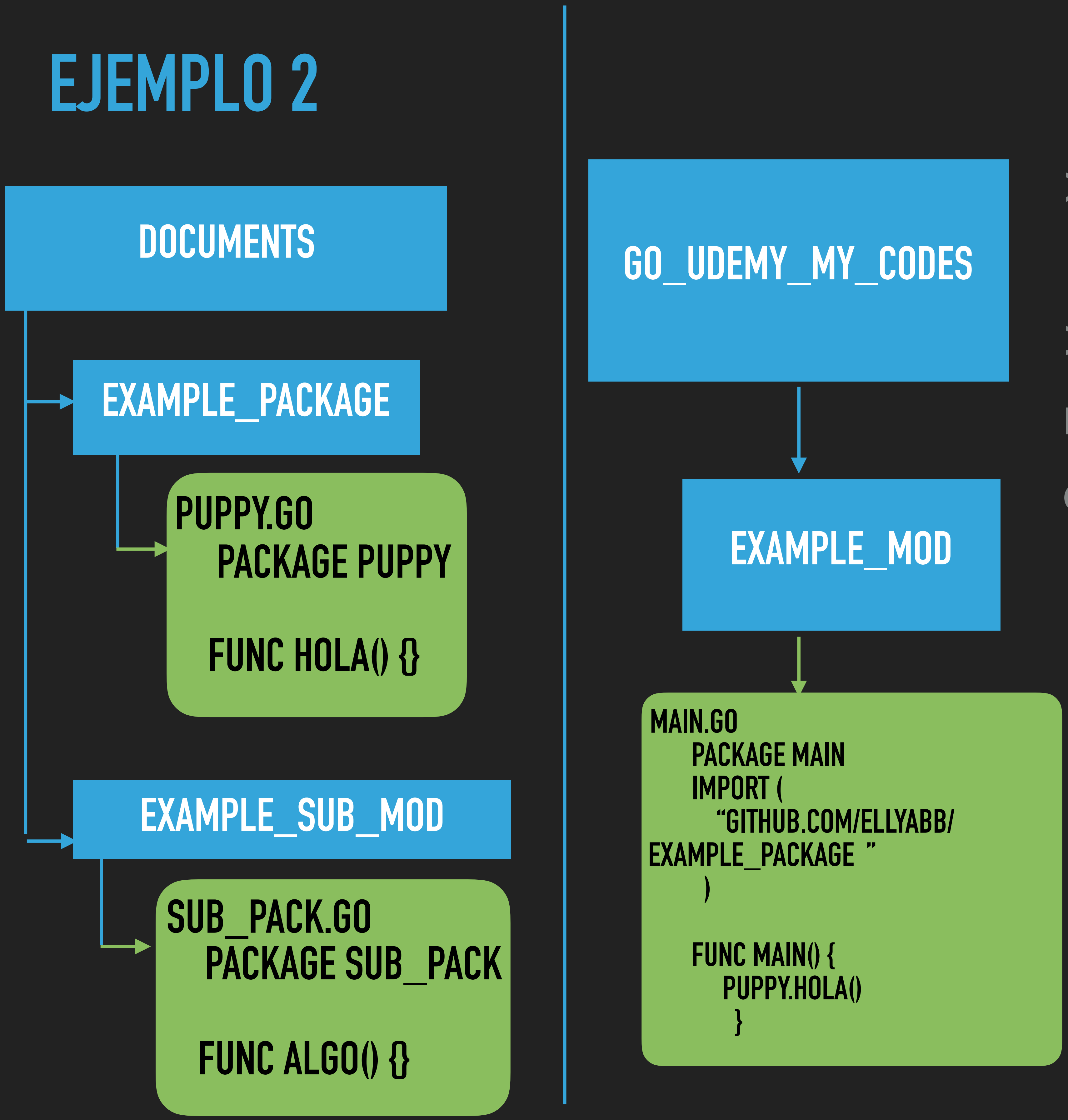
>go mod init **mycodes**

>go get GitHub.com/name/example@#commit

VERSION

- go mod init ______(name)

EJEMPLO 2



```
PUPPY.GO
    PACKAGE PUPPY

   FUNC HOLA() {}
```

```
SUB_PACK.GO
    PACKAGE SUB_PACK

   FUNC ALGO() {}
```

```
MAIN.GO
     PACKAGE MAIN
     IMPORT (
        "GITHUB.COM/ELLYABB/
EXAMPLE_PACKAGE "
       )

     FUNC MAIN() {
         PUPPY.HOLA()
          }
```

>go mod init **mycodes**

>go get GitHub.com/name/example@#commit