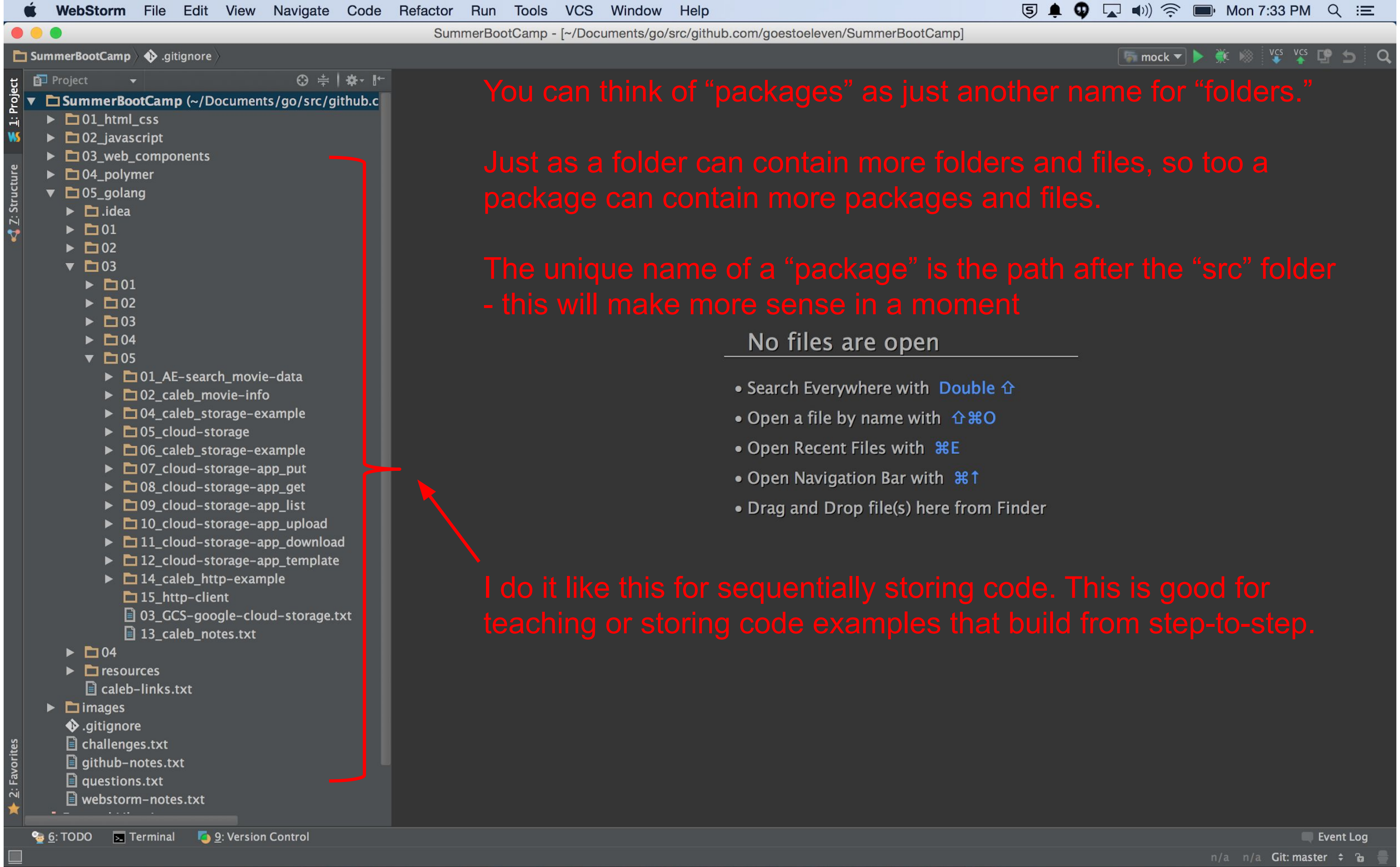
Packages && Docs

unique namespaces


WebStorm File Edit View Navigate Code Refactor Run Tools VCS Window Help
Mon 7:33 PM
SummerBootCamp - [~/Documents/go/src/github.com/goestoeleven/SummerBootCamp]
SummerBootCamp
.gitignore
mock
Project
1: Project
7: Structure
2: Favorites
6: TODO
Terminal
9: Version Control
Event Log
n/a n/a Git: master

You can think of "packages" as just another name for "folders."

Just as a folder can contain more folders and files, so too a package can contain more packages and files.

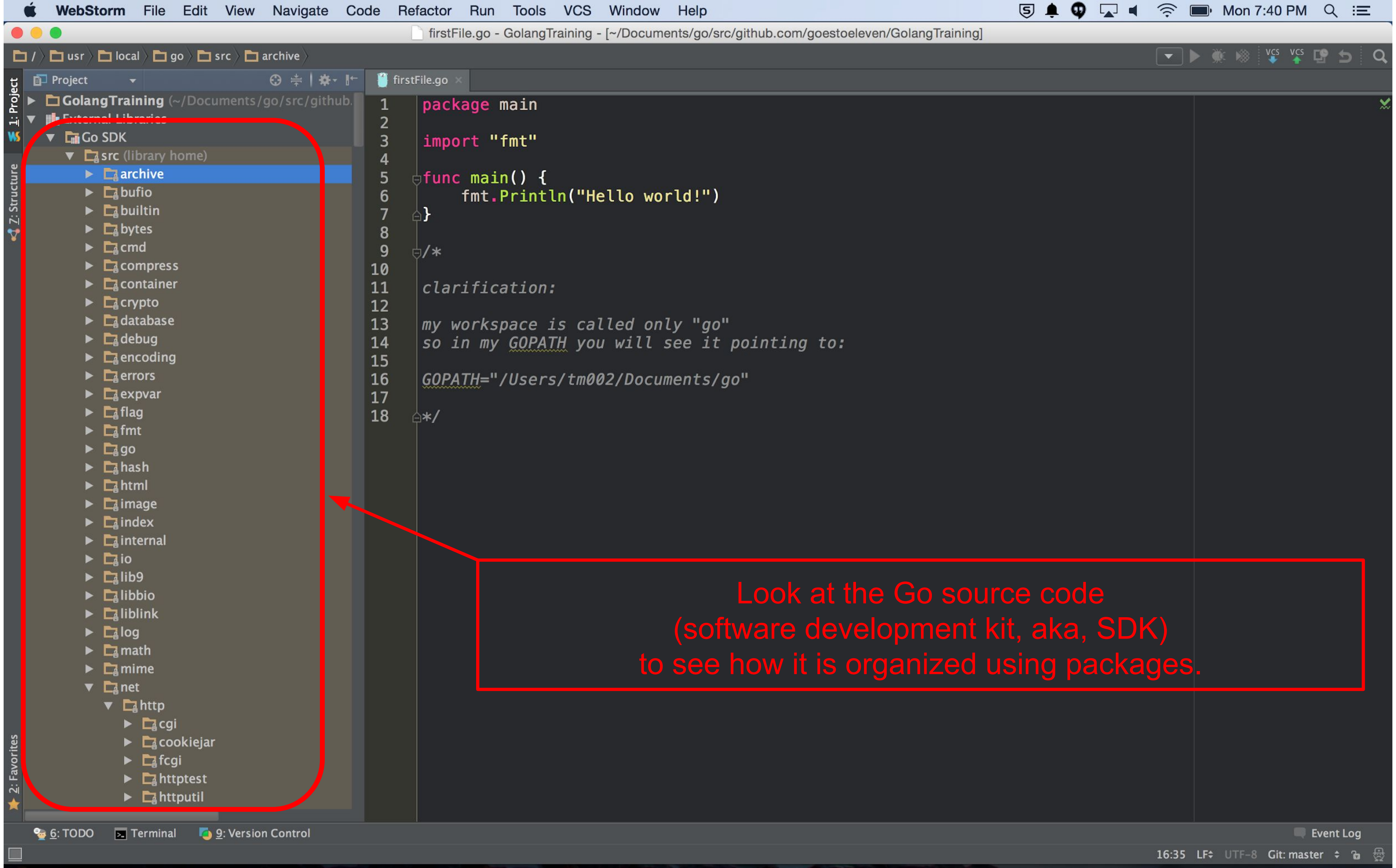
The unique name of a "package" is the path after the "src" folder - this will make more sense in a moment

```
SummerBootCamp (~/Documents/go/src/github.c
  01_html_css
  02_javascript
  03_web_components
  04_polymer
  05_golang
    .idea
    01
    02
    03
      01
      02
      03
      04
      05
        01_AE-search_movie-data
        02_caleb_movie-info
        04_caleb_storage-example
        05_cloud-storage
        06_caleb_storage-example
        07_cloud-storage-app_put
        08_cloud-storage-app_get
        09_cloud-storage-app_list
        10_cloud-storage-app_upload
        11_cloud-storage-app_download
        12_cloud-storage-app_template
        14_caleb_http-example
        15_http-client
        03_GCS-google-cloud-storage.txt
        13_caleb_notes.txt
    04
    resources
    caleb-links.txt
  images
  .gitignore
  challenges.txt
  github-notes.txt
  questions.txt
  webstorm-notes.txt
```

No files are open

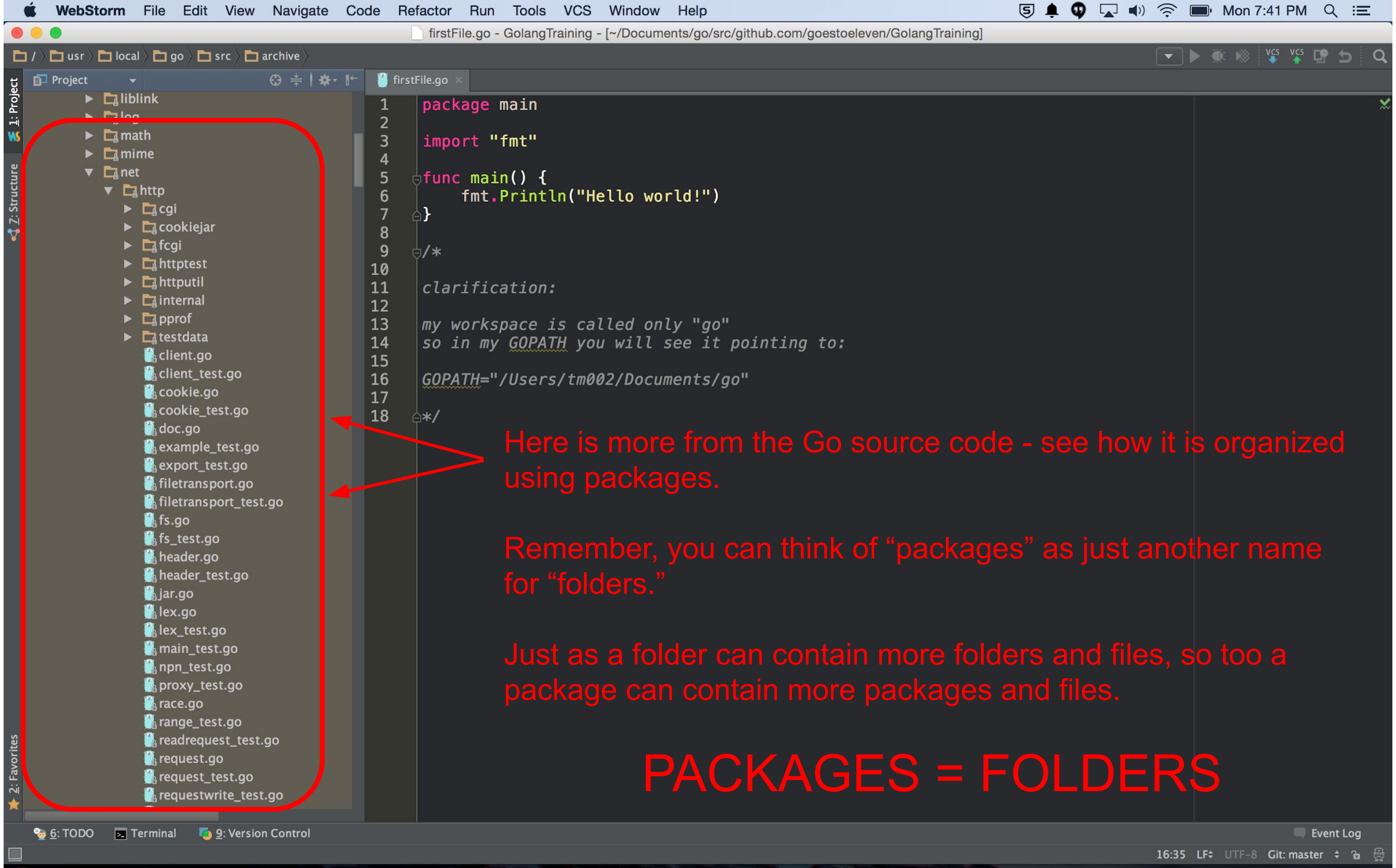
- Search Everywhere with Double ⇧
- Open a file by name with ⇧⌘O
- Open Recent Files with ⌘E
- Open Navigation Bar with ⌘↑
- Drag and Drop file(s) here from Finder

I do it like this for sequentially storing code. This is good for teaching or storing code examples that build from step-to-step.



```go
package main

import "fmt"

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

/*

clarification:

my workspace is called only "go"
so in my GOPATH you will see it pointing to:

GOPATH="/Users/tm002/Documents/go"

*/
```

Look at the Go source code
(software development kit, aka, SDK)
to see how it is organized using packages.



```go
package main

import "fmt"

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

/*

clarification:

my workspace is called only "go"
so in my GOPATH you will see it pointing to:

GOPATH="/Users/tm002/Documents/go"

*/
```

Here is more from the Go source code - see how it is organized using packages.

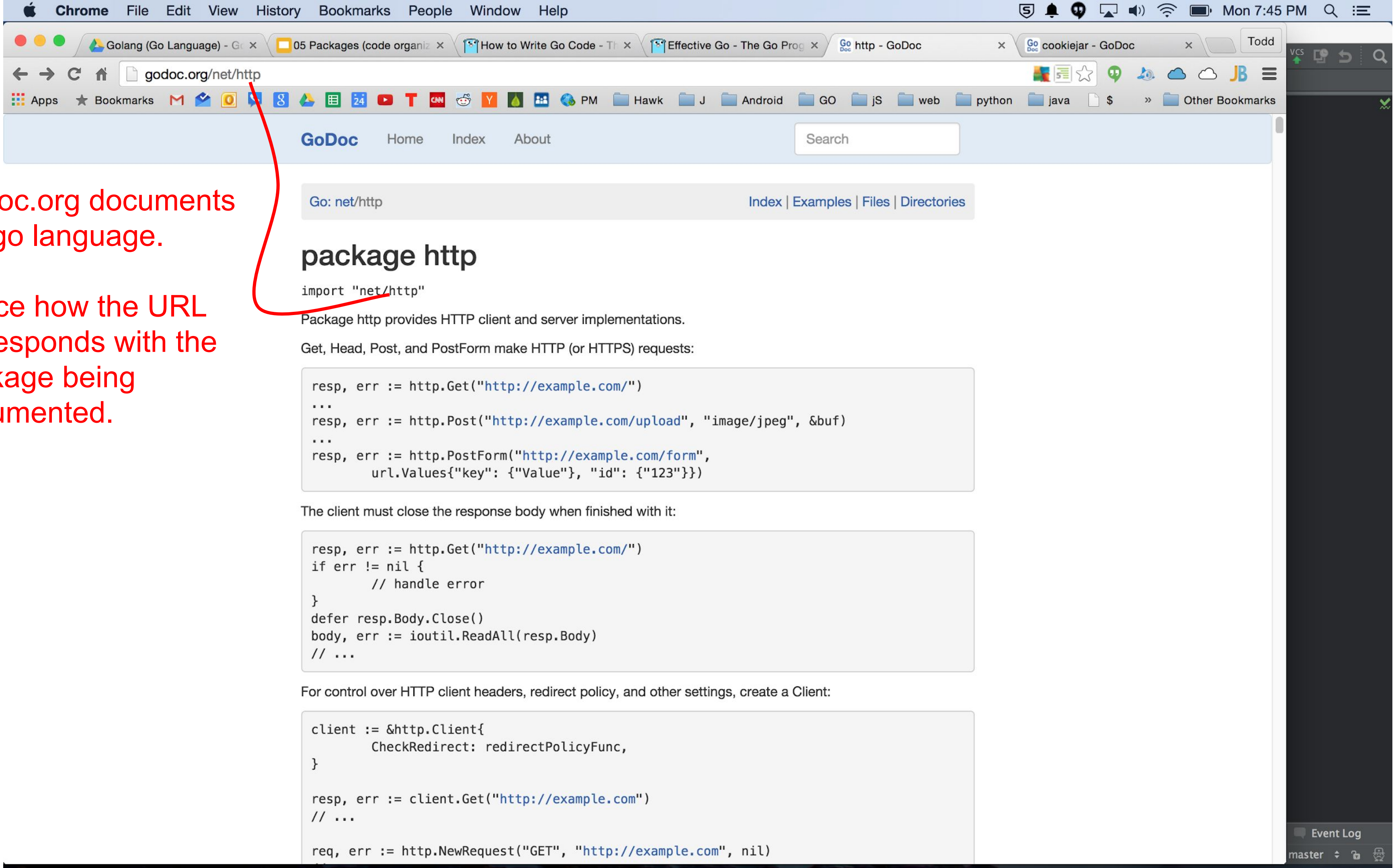
Remember, you can think of “packages” as just another name for “folders.”

Just as a folder can contain more folders and files, so too a package can contain more packages and files.

PACKAGES = FOLDERS

Godoc.org documents the go language.

Notice how the URL corresponds with the package being documented.



package http

```
import "net/http"
```

Package http provides HTTP client and server implementations.

Get, Head, Post, and PostForm make HTTP (or HTTPS) requests:

```
resp, err := http.Get("http://example.com/")
...
resp, err := http.Post("http://example.com/upload", "image/jpeg", &buf)
...
resp, err := http.PostForm("http://example.com/form",
        url.Values{"key": {"Value"}, "id": {"123"}})
```

The client must close the response body when finished with it:

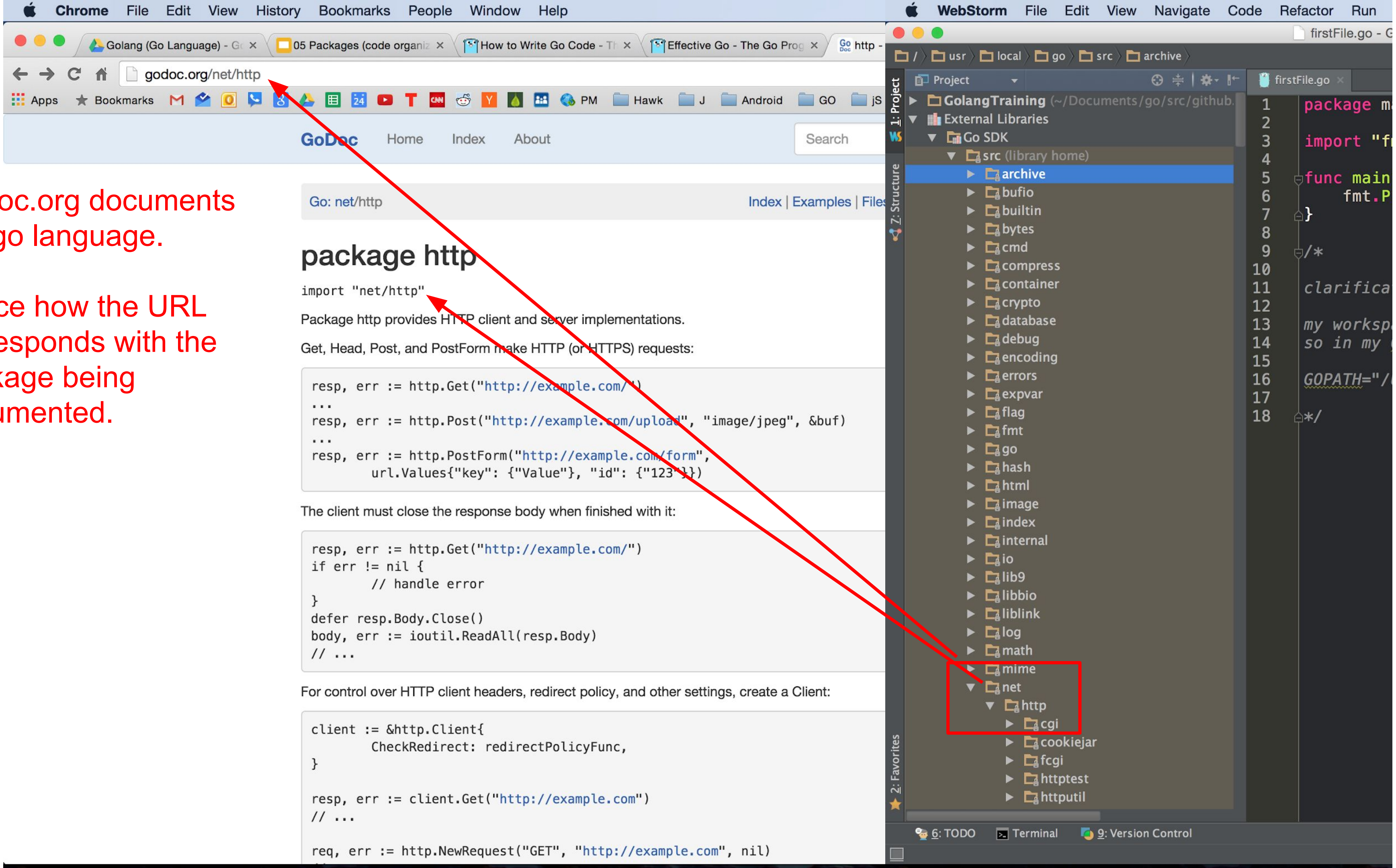
```
resp, err := http.Get("http://example.com/")
if err != nil {
        // handle error
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
// ...
```

For control over HTTP client headers, redirect policy, and other settings, create a Client:

```
client := &http.Client{
        CheckRedirect: redirectPolicyFunc,
}

resp, err := client.Get("http://example.com")
// ...

req, err := http.NewRequest("GET", "http://example.com", nil)
```

Godoc.org documents the go language.

Notice how the URL corresponds with the package being documented.



Go: net/http

Index | Examples | Files

package http

```
import "net/http"
```

Package http provides HTTP client and server implementations.

Get, Head, Post, and PostForm make HTTP (or HTTPS) requests:

```
resp, err := http.Get("http://example.com/")
...
resp, err := http.Post("http://example.com/upload", "image/jpeg", &buf)
...
resp, err := http.PostForm("http://example.com/form",
        url.Values{"key": {"Value"}, "id": {"123"}})
```

The client must close the response body when finished with it:

```
resp, err := http.Get("http://example.com/")
if err != nil {
        // handle error
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
// ...
```

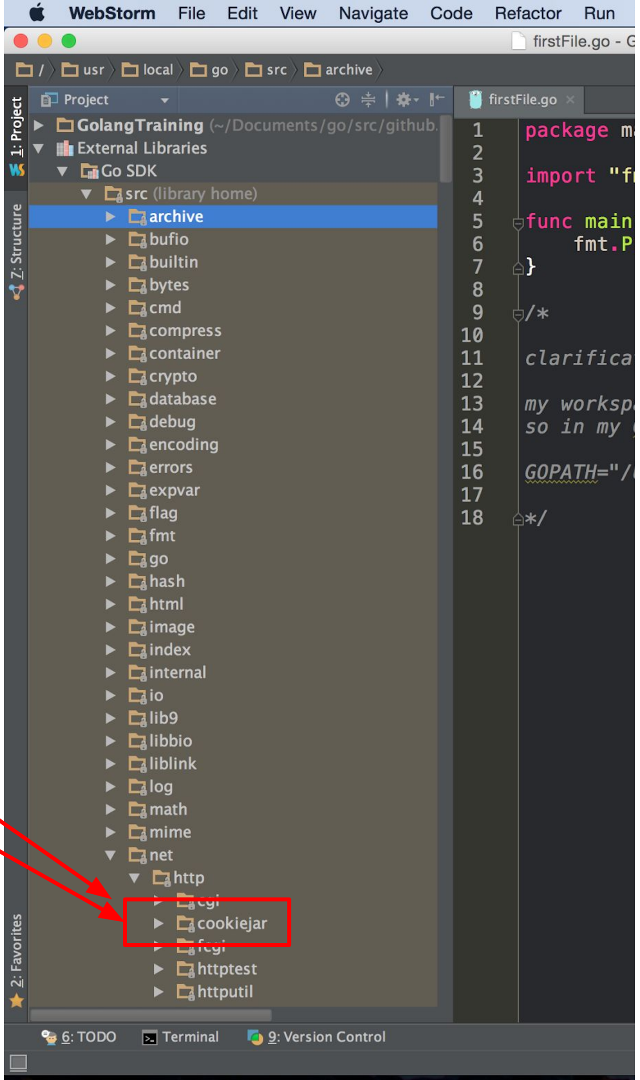
For control over HTTP client headers, redirect policy, and other settings, create a Client:

```
client := &http.Client{
        CheckRedirect: redirectPolicyFunc,
}

resp, err := client.Get("http://example.com")
// ...

req, err := http.NewRequest("GET", "http://example.com", nil)
```

```
package m

import "f

func main
    fmt.P
}

/*

clarifica

my worksp
so in my

GOPATH="/

*/
```

What is this package?

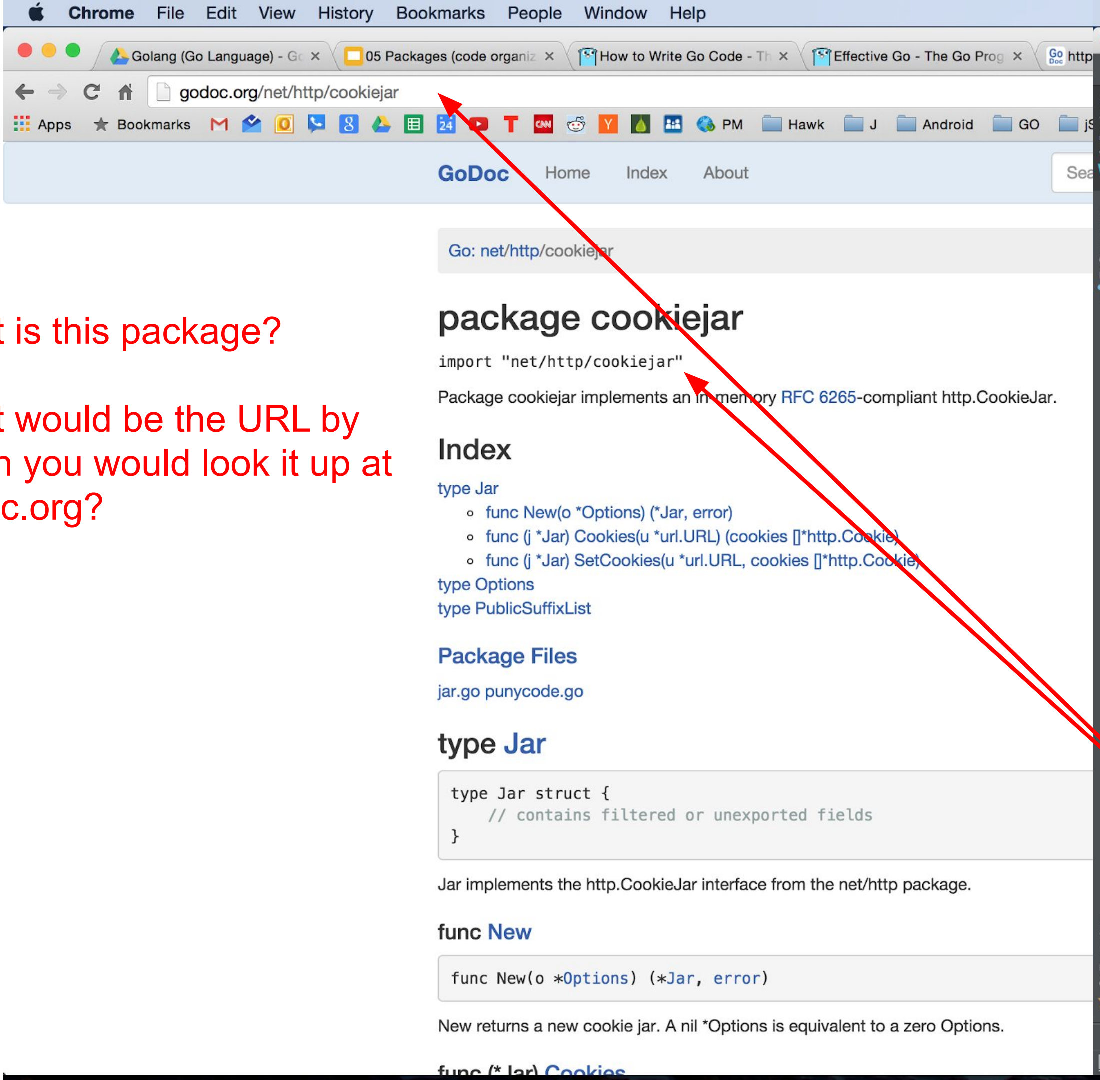
What would be the URL by which you would look it up at godoc.org?



```
package m
import "f
func main
    fmt.P
}
/*
clarifica
my worksp
so in my
GOPATH="/
*/
```

What is this package?

What would be the URL by which you would look it up at godoc.org?


Chrome File Edit View History Bookmarks People Window Help
Golang (Go Language) - Go
05 Packages (code organiz
How to Write Go Code - Th
Effective Go - The Go Prog
godoc.org/net/http/cookiejar
Apps Bookmarks PM Hawk J Android GO
GoDoc Home Index About

Go: net/http/cookiejar

package cookiejar

```
import "net/http/cookiejar"
```

Package cookiejar implements an in-memory RFC 6265-compliant http.CookieJar.

Index

type Jar
- func New(o *Options) (*Jar, error)
- func (j *Jar) Cookies(u *url.URL) (cookies []*http.Cookie)
- func (j *Jar) SetCookies(u *url.URL, cookies []*http.Cookie)

type Options
type PublicSuffixList

Package Files

jar.go punycode.go

type Jar

```
type Jar struct {
    // contains filtered or unexported fields
}
```

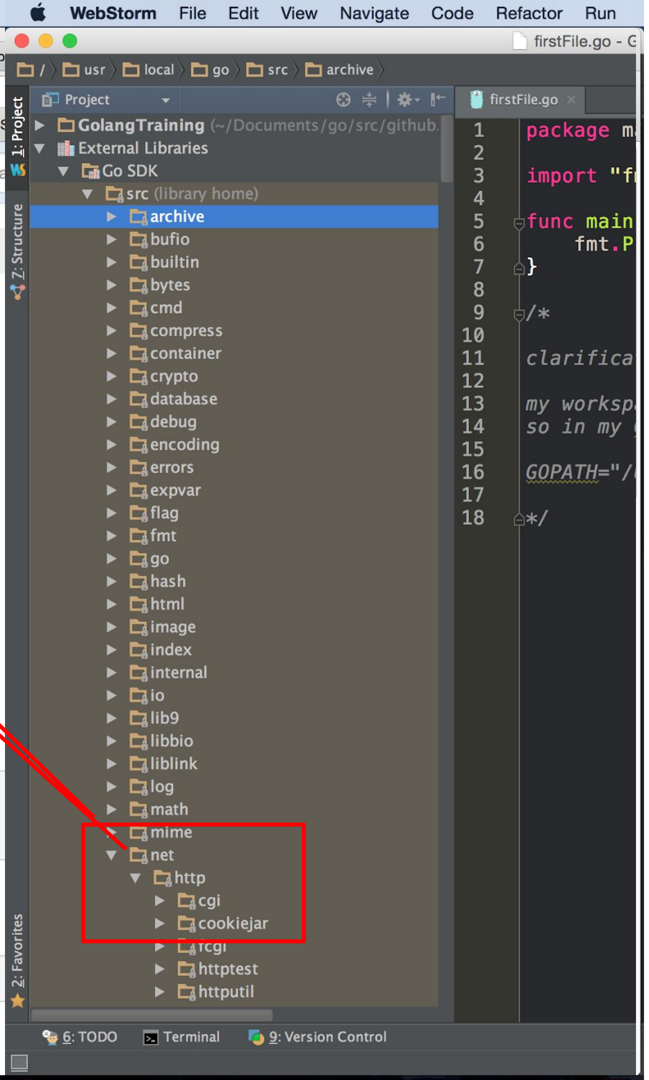
Jar implements the http.CookieJar interface from the net/http package.

func New

```
func New(o *Options) (*Jar, error)
```

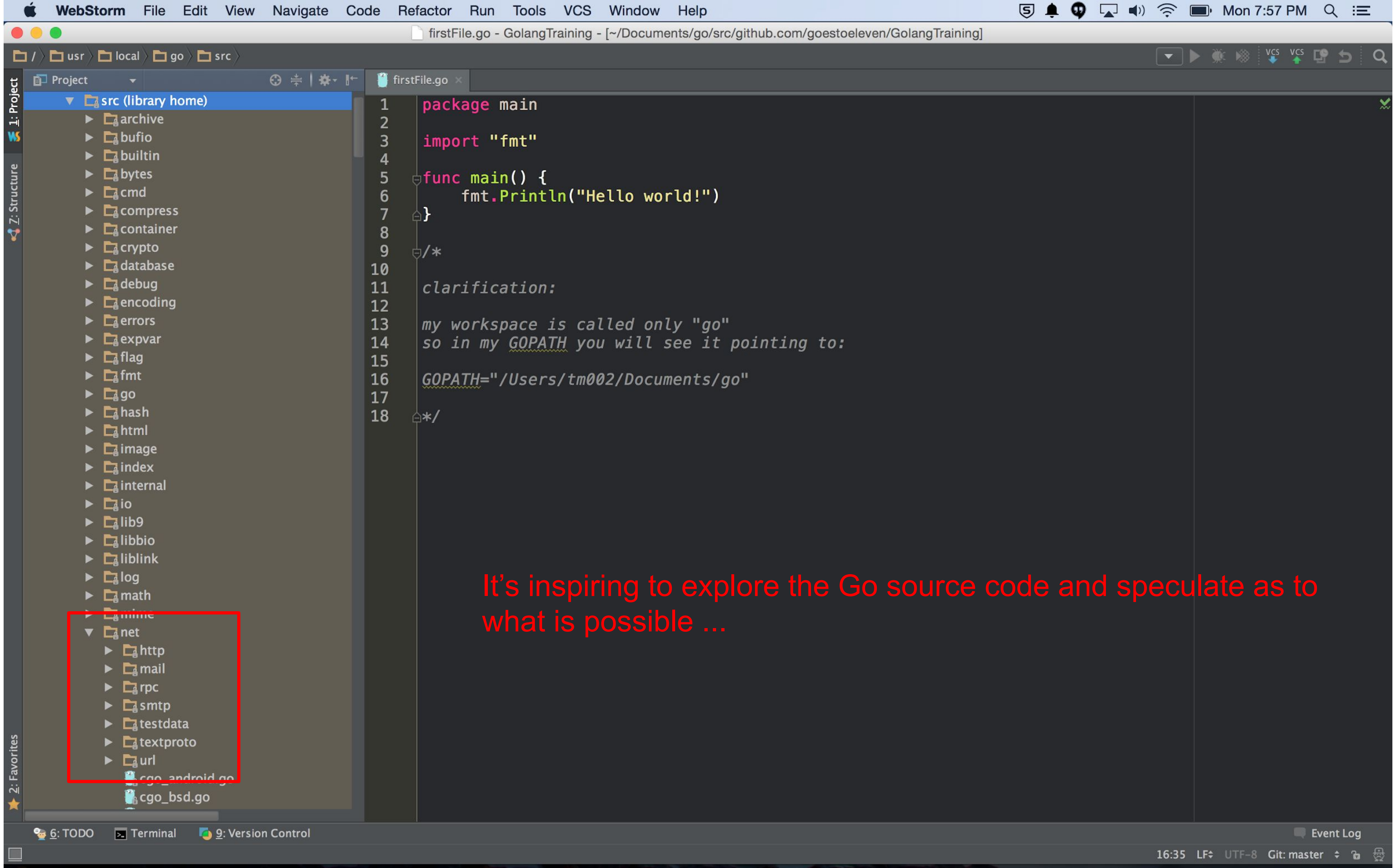
New returns a new cookie jar. A nil *Options is equivalent to a zero Options.

func (*Jar) Cookies


WebStorm File Edit View Navigate Code Refactor Run
firstFile.go - G
/ usr local go src archive
Project
firstFile.go
1: Project
7: Structure
2: Favorites
6: TODO Terminal 9: Version Control

```
GolangTraining (~/Documents/go/src/github.
External Libraries
  Go SDK
    src (library home)
      archive
      bufio
      builtin
      bytes
      cmd
      compress
      container
      crypto
      database
      debug
      encoding
      errors
      expvar
      flag
      fmt
      go
      hash
      html
      image
      index
      internal
      io
      lib9
      libbio
      liblink
      log
      math
      mime
      net
        http
          cgi
          cookiejar
          fcgi
          httptest
          httputil
```

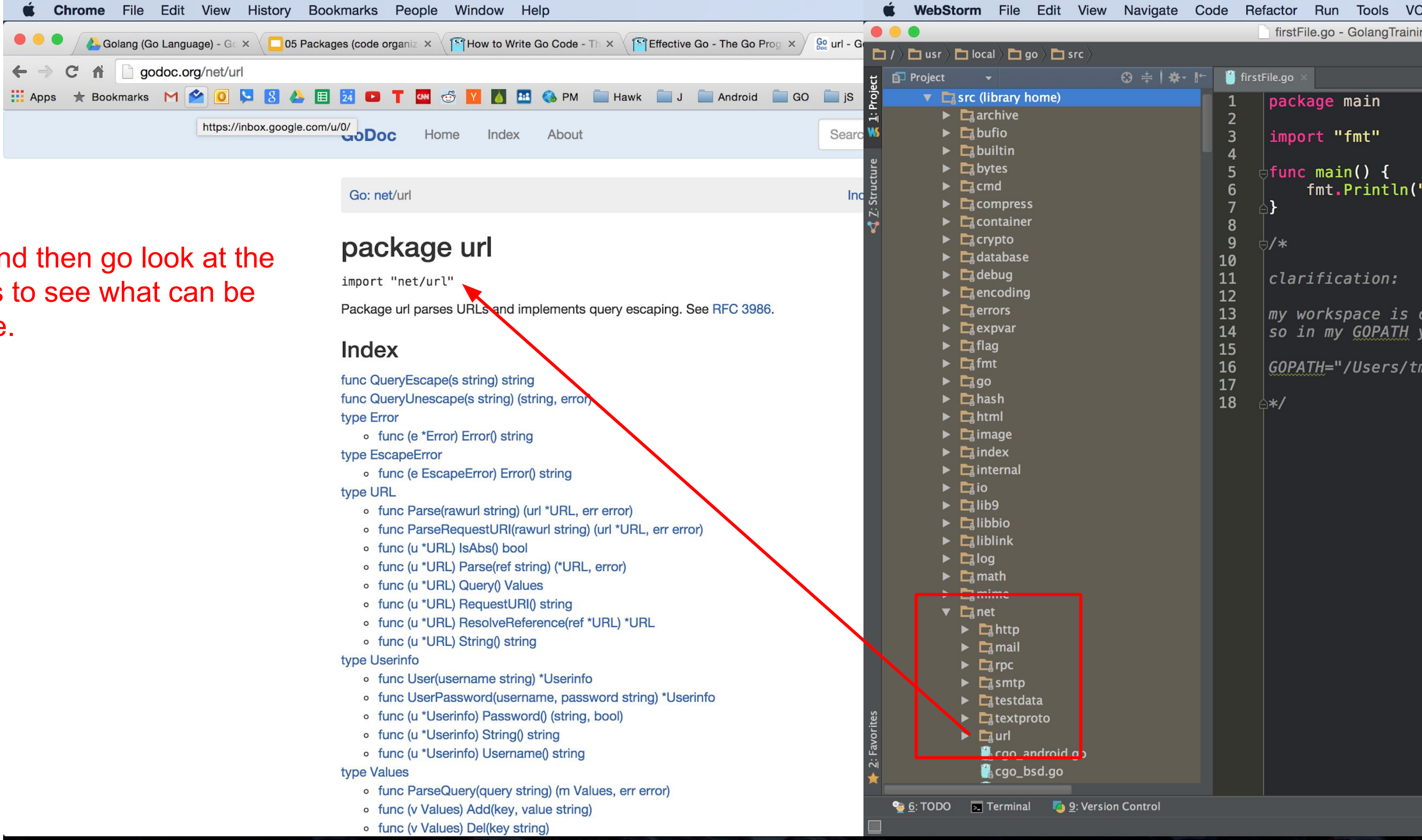
```
package m

import "f

func main
    fmt.P
}

/*

clarifica

my worksp
so in my

GOPATH="/

*/
```



```go
package main

import "fmt"

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

/*

clarification:

my workspace is called only "go"
so in my GOPATH you will see it pointing to:

GOPATH="/Users/tm002/Documents/go"

*/
```

It's inspiring to explore the Go source code and speculate as to what is possible ...

… and then go look at the docs to see what can be done.


Chrome File Edit View History Bookmarks People Window Help
Golang (Go Language) - G
05 Packages (code organiz
How to Write Go Code - Th
Effective Go - The Go Prog
url - G
godoc.org/net/url
Apps Bookmarks PM Hawk J Android GO jS
https://inbox.google.com/u/0/
GoDoc Home Index About Searc
WebStorm File Edit View Navigate Code Refactor Run Tools VC
firstFile.go - GolangTrainin
/ usr local go src
Project
firstFile.go
1: Project
7: Structure
2: Favorites
6: TODO
Terminal
9: Version Control

Go: net/url

package url

```
import "net/url"
```

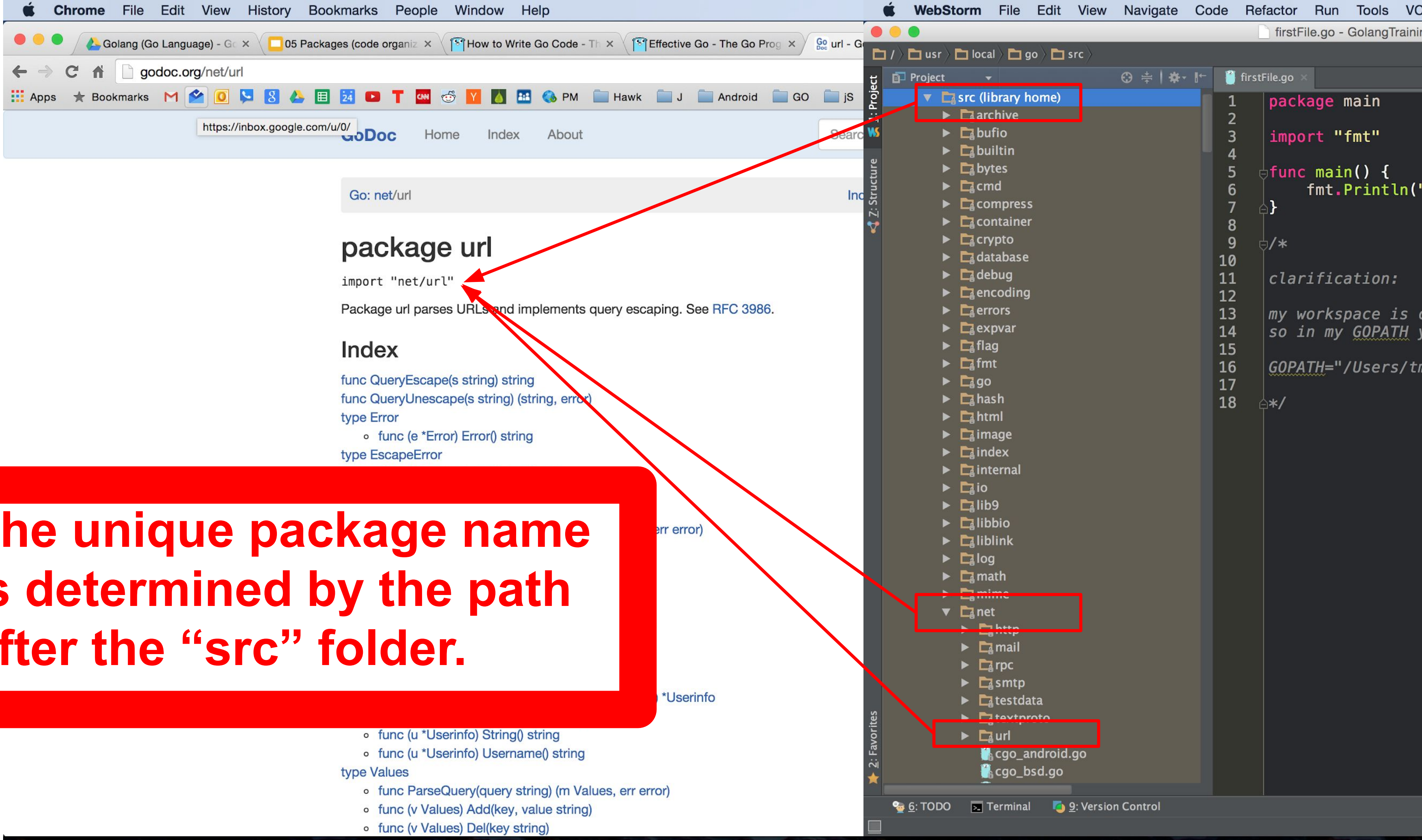
Package url parses URLs and implements query escaping. See RFC 3986.

Index

- func QueryEscape(s string) string
- func QueryUnescape(s string) (string, error)
- type Error
  - func (e *Error) Error() string
- type EscapeError
  - func (e EscapeError) Error() string
- type URL
  - func Parse(rawurl string) (url *URL, err error)
  - func ParseRequestURI(rawurl string) (url *URL, err error)
  - func (u *URL) IsAbs() bool
  - func (u *URL) Parse(ref string) (*URL, error)
  - func (u *URL) Query() Values
  - func (u *URL) RequestURI() string
  - func (u *URL) ResolveReference(ref *URL) *URL
  - func (u *URL) String() string
- type Userinfo
  - func User(username string) *Userinfo
  - func UserPassword(username, password string) *Userinfo
  - func (u *Userinfo) Password() (string, bool)
  - func (u *Userinfo) String() string
  - func (u *Userinfo) Username() string
- type Values
  - func ParseQuery(query string) (m Values, err error)
  - func (v Values) Add(key, value string)
  - func (v Values) Del(key string)

```
src (library home)
  archive
  bufio
  builtin
  bytes
  cmd
  compress
  container
  crypto
  database
  debug
  encoding
  errors
  expvar
  flag
  fmt
  go
  hash
  html
  image
  index
  internal
  io
  lib9
  libbio
  liblink
  log
  math
  mime
  net
    http
    mail
    rpc
    smtp
    testdata
    textproto
    url
    cgo_android.go
    cgo_bsd.go
```

```go
package main

import "fmt"

func main() {
	fmt.Println('
}

/*

clarification:

my workspace is
so in my GOPATH

GOPATH="/Users/t

*/
```



godoc.org/net/url

Go: net/url

package url

```
import "net/url"
```

Package url parses URLs and implements query escaping. See RFC 3986.

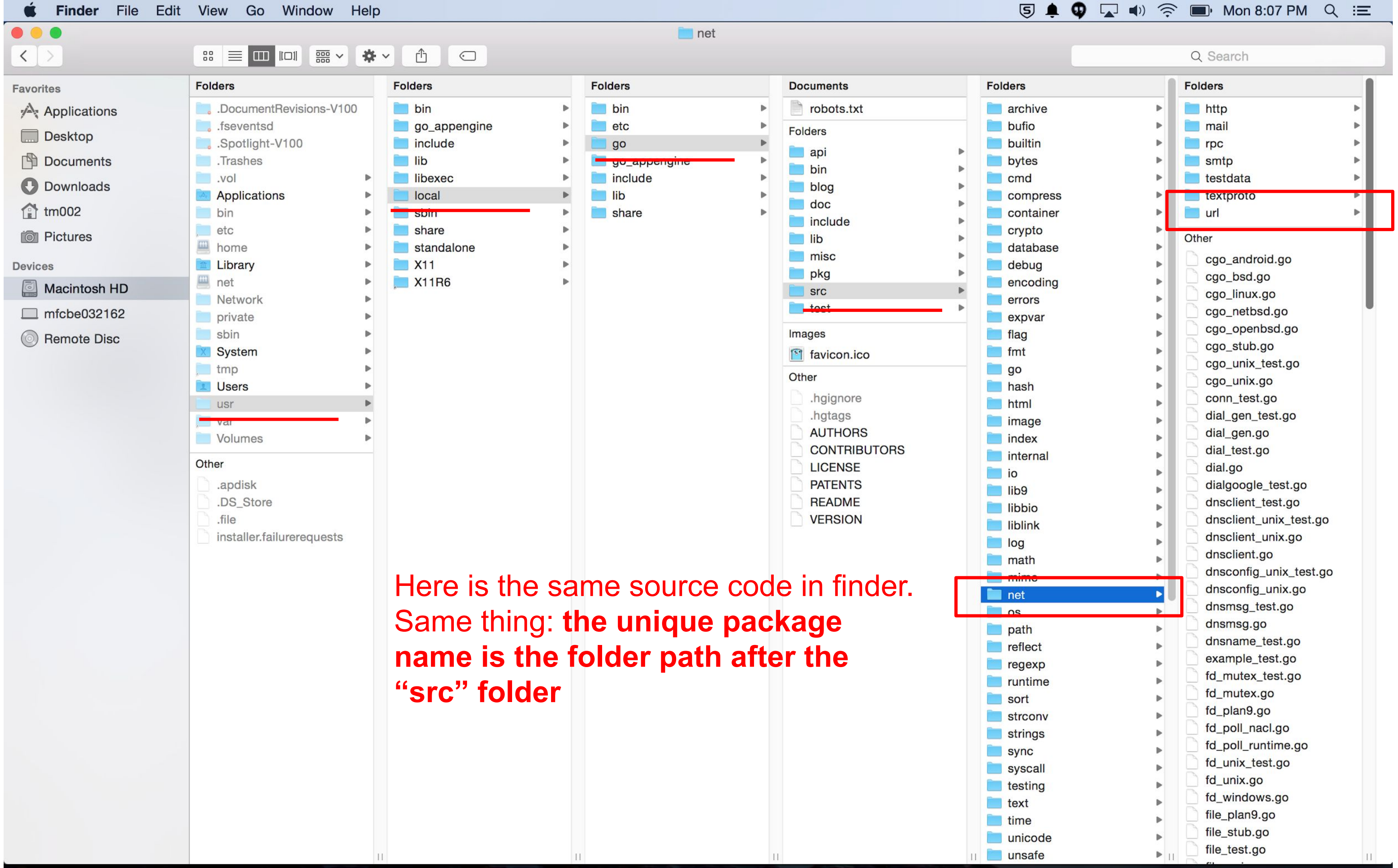
Index

func QueryEscape(s string) string
func QueryUnescape(s string) (string, error)
type Error
- func (e *Error) Error() string
type EscapeError

- func (u *Userinfo) String() string
- func (u *Userinfo) Username() string
type Values
- func ParseQuery(query string) (m Values, err error)
- func (v Values) Add(key, value string)
- func (v Values) Del(key string)

```
package main

import "fmt"

func main() {
	fmt.Println('
}

/*

clarification:

my workspace is c
so in my GOPATH y

GOPATH="/Users/tr

*/
```

The unique package name is determined by the path after the “src” folder.



Folders
.DocumentRevisions-V100
.fseventsd
.Spotlight-V100
.Trashes
.vol
Applications
bin
etc
home
Library
net
Network
private
sbin
System
tmp
Users
usr
~~var~~
Volumes
Other
.apdisk
.DS_Store
.file
installer.failurerequests

Folders
bin
go_appengine
include
lib
libexec
local
~~sbin~~
share
standalone
X11
X11R6

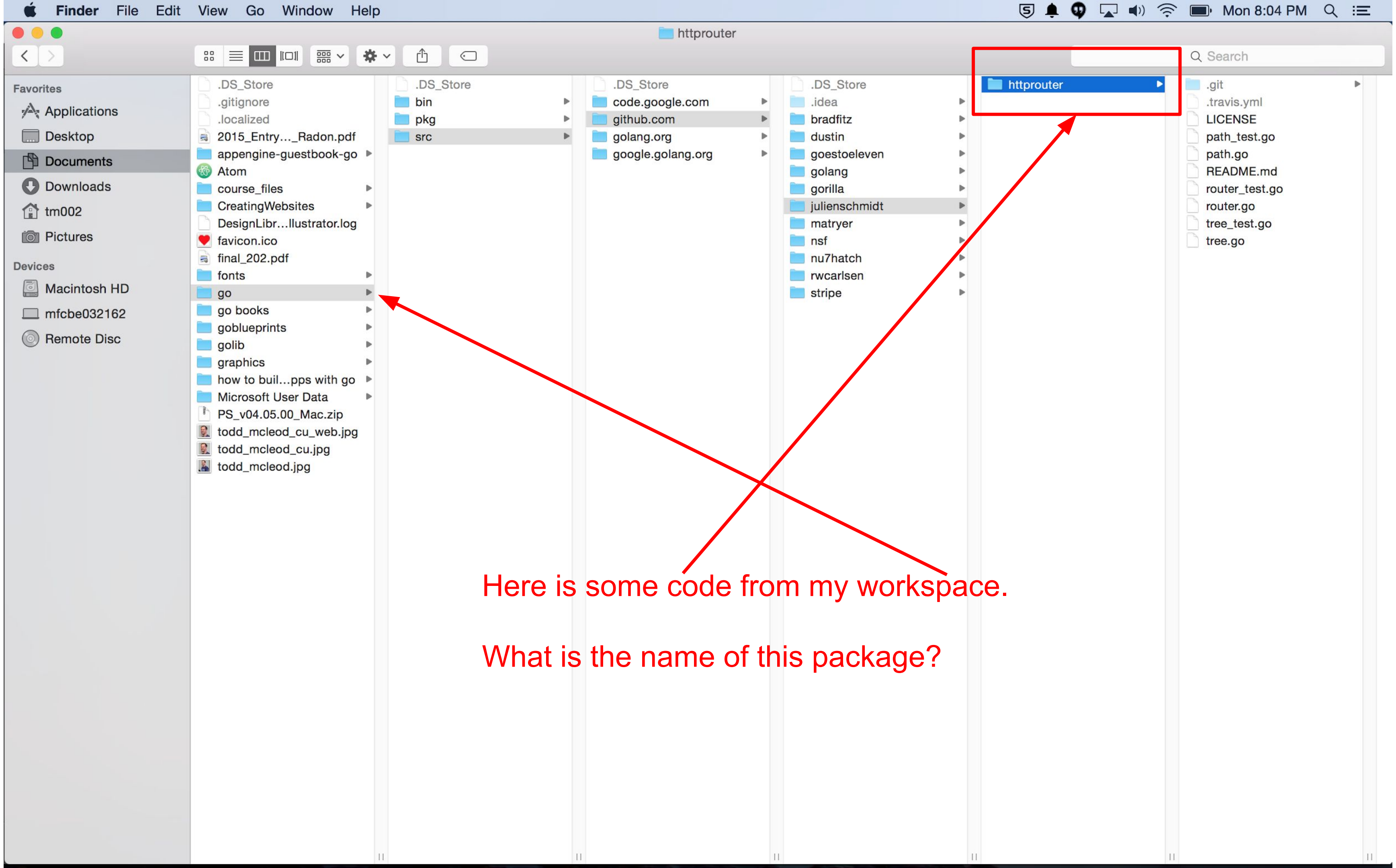
Folders
bin
etc
go
~~go_appengine~~
include
lib
share

Documents
robots.txt
Folders
api
bin
blog
doc
include
lib
misc
pkg
src
~~test~~
Images
favicon.ico
Other
.hgignore
.hgtags
AUTHORS
CONTRIBUTORS
LICENSE
PATENTS
README
VERSION

Folders
archive
bufio
builtin
bytes
cmd
compress
container
crypto
database
debug
encoding
errors
expvar
flag
fmt
go
hash
html
image
index
internal
io
lib9
libbio
liblink
log
math
mime
net
os
path
reflect
regexp
runtime
sort
strconv
strings
sync
syscall
testing
text
time
unicode
unsafe

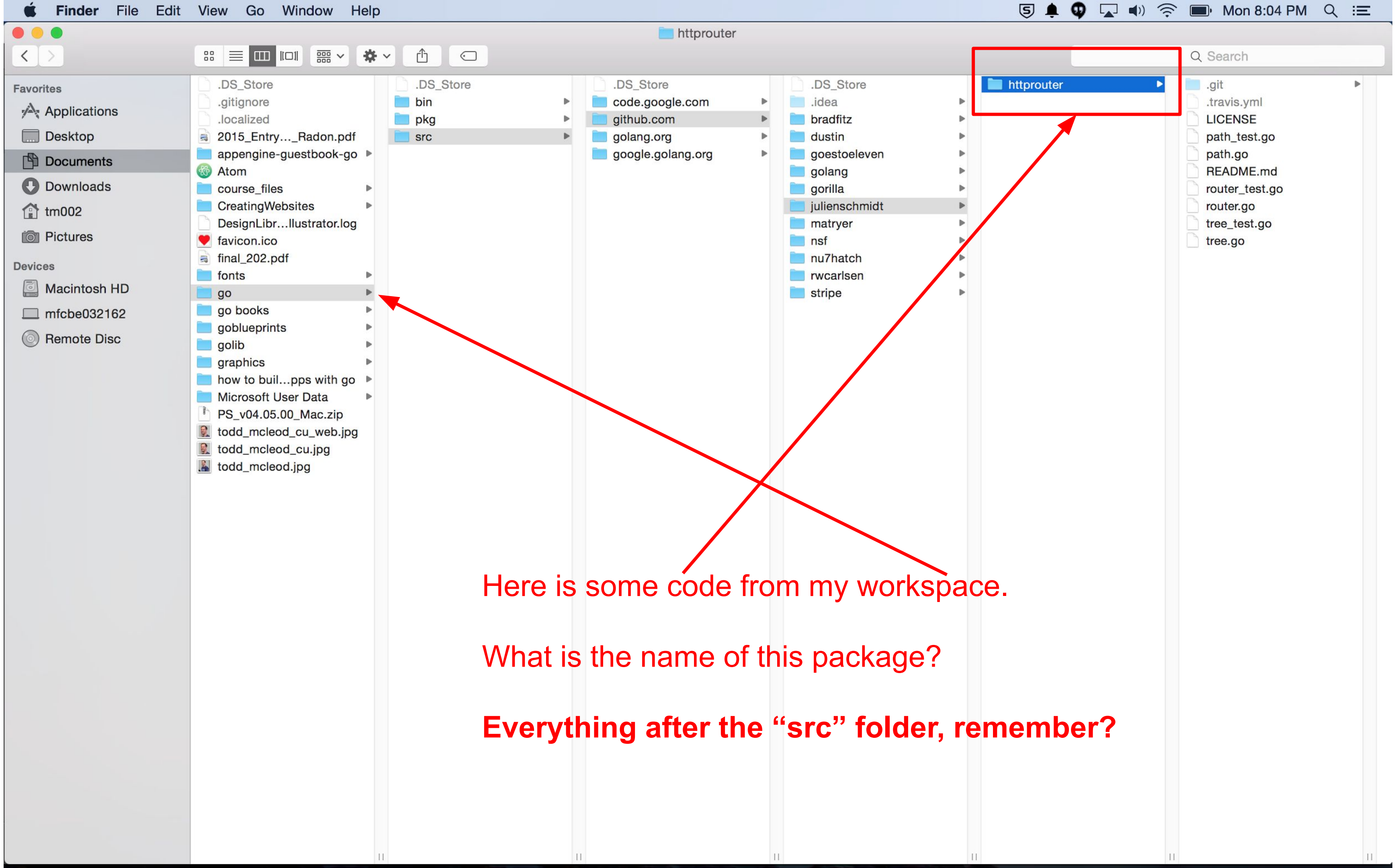
Folders
http
mail
rpc
smtp
testdata
textproto
url
Other
cgo_android.go
cgo_bsd.go
cgo_linux.go
cgo_netbsd.go
cgo_openbsd.go
cgo_stub.go
cgo_unix_test.go
cgo_unix.go
conn_test.go
dial_gen_test.go
dial_gen.go
dial_test.go
dial.go
dialgoogle_test.go
dnsclient_test.go
dnsclient_unix_test.go
dnsclient_unix.go
dnsclient.go
dnsconfig_unix_test.go
dnsconfig_unix.go
dnsmsg_test.go
dnsmsg.go
dnsname_test.go
example_test.go
fd_mutex_test.go
fd_mutex.go
fd_plan9.go
fd_poll_nacl.go
fd_poll_runtime.go
fd_unix_test.go
fd_unix.go
fd_windows.go
file_plan9.go
file_stub.go
file_test.go

Here is the same source code in finder. Same thing: **the unique package name is the folder path after the "src" folder**



Here is some code from my workspace.

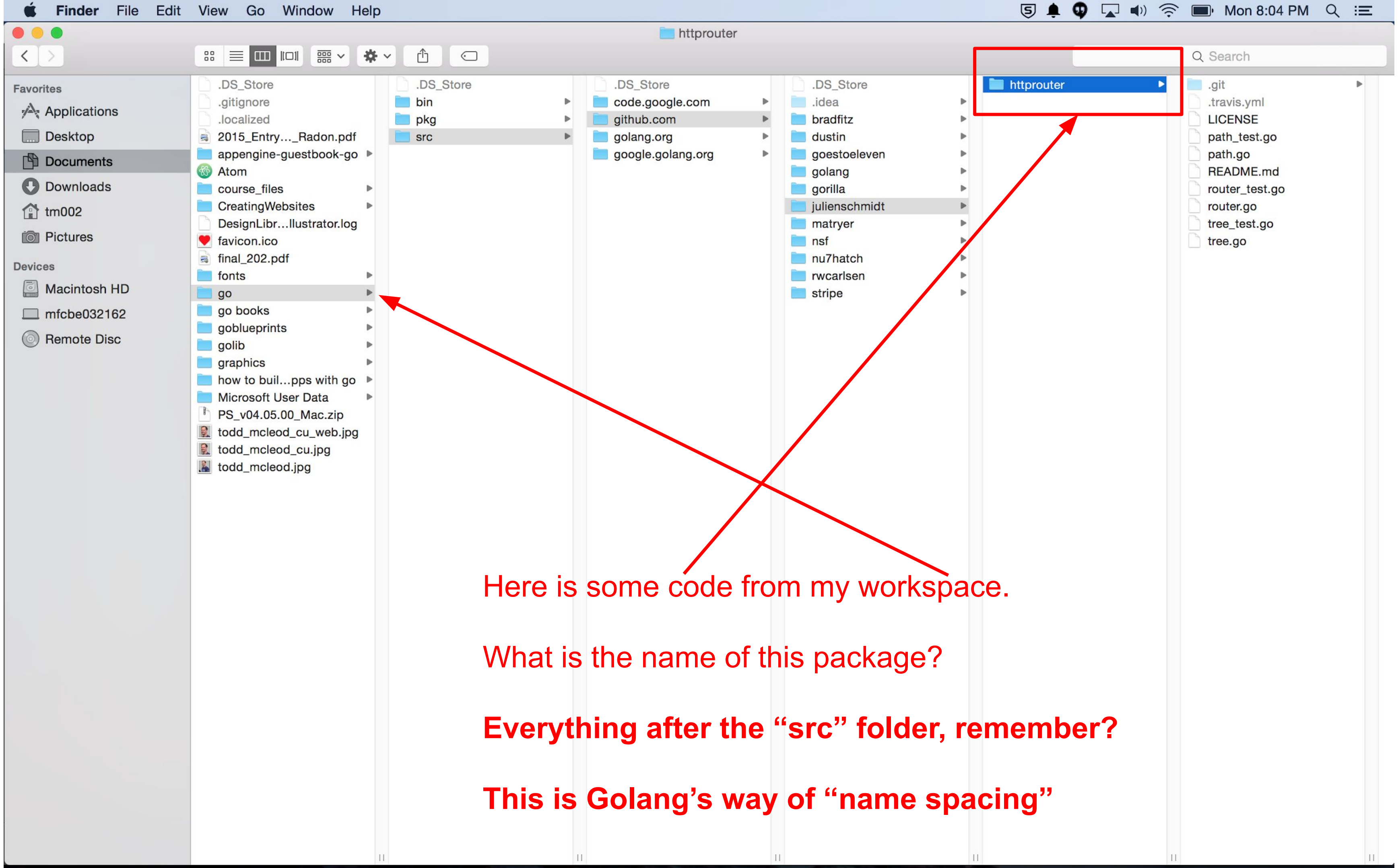
What is the name of this package?



Here is some code from my workspace.

What is the name of this package?

Everything after the "src" folder, remember?



Here is some code from my workspace.

What is the name of this package?

Everything after the “src” folder, remember?

This is Golang’s way of “name spacing”

namespace

In computing, a **namespace** is used to organize objects of various kinds, so that these objects may be referred to by name.

Examples include:

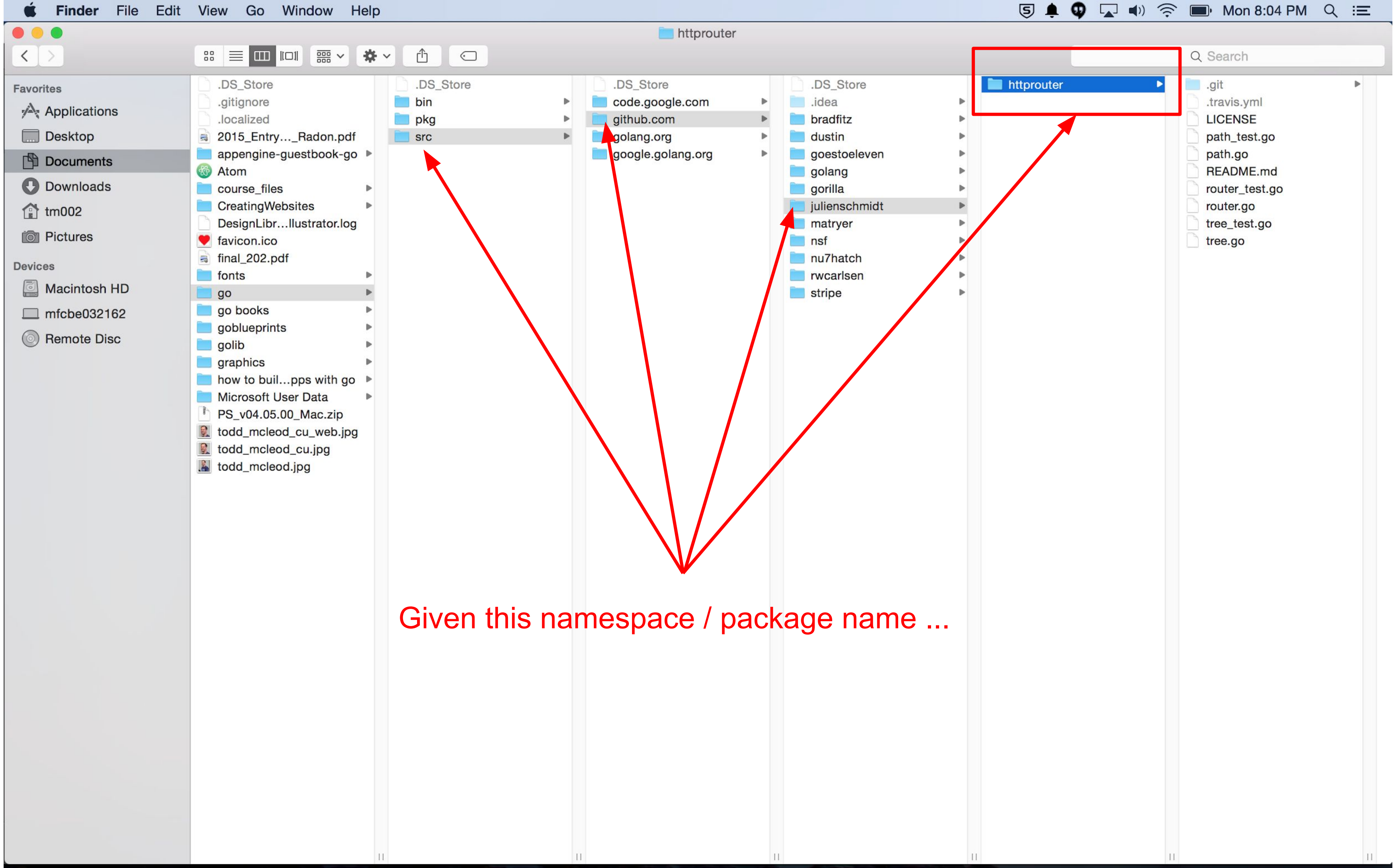
- file systems are **namespaces** that assign names to files
- programming languages organize variables and subroutines in **namespaces**
- computer networks and distributed systems assign names to resources, such as computers, printers, websites, (remote) files, etc.

Namespaces are commonly structured as hierarchies to allow reuse of names in different contexts.

Example: human names. Jane Doe. Within the **namespace** of the Doe family, just "Jane" suffices to unambiguously designate this person, while within the "global" **namespace** of all people, the full name must be used.

In a similar way, hierarchical file systems organize files in directories. Each directory is a separate **namespace**, so that the directories may both contain a file "jane". In computer programming, **namespaces** are typically employed *for the purpose of avoiding name collisions between multiple identifiers* that share the same name.

source: wikipedia

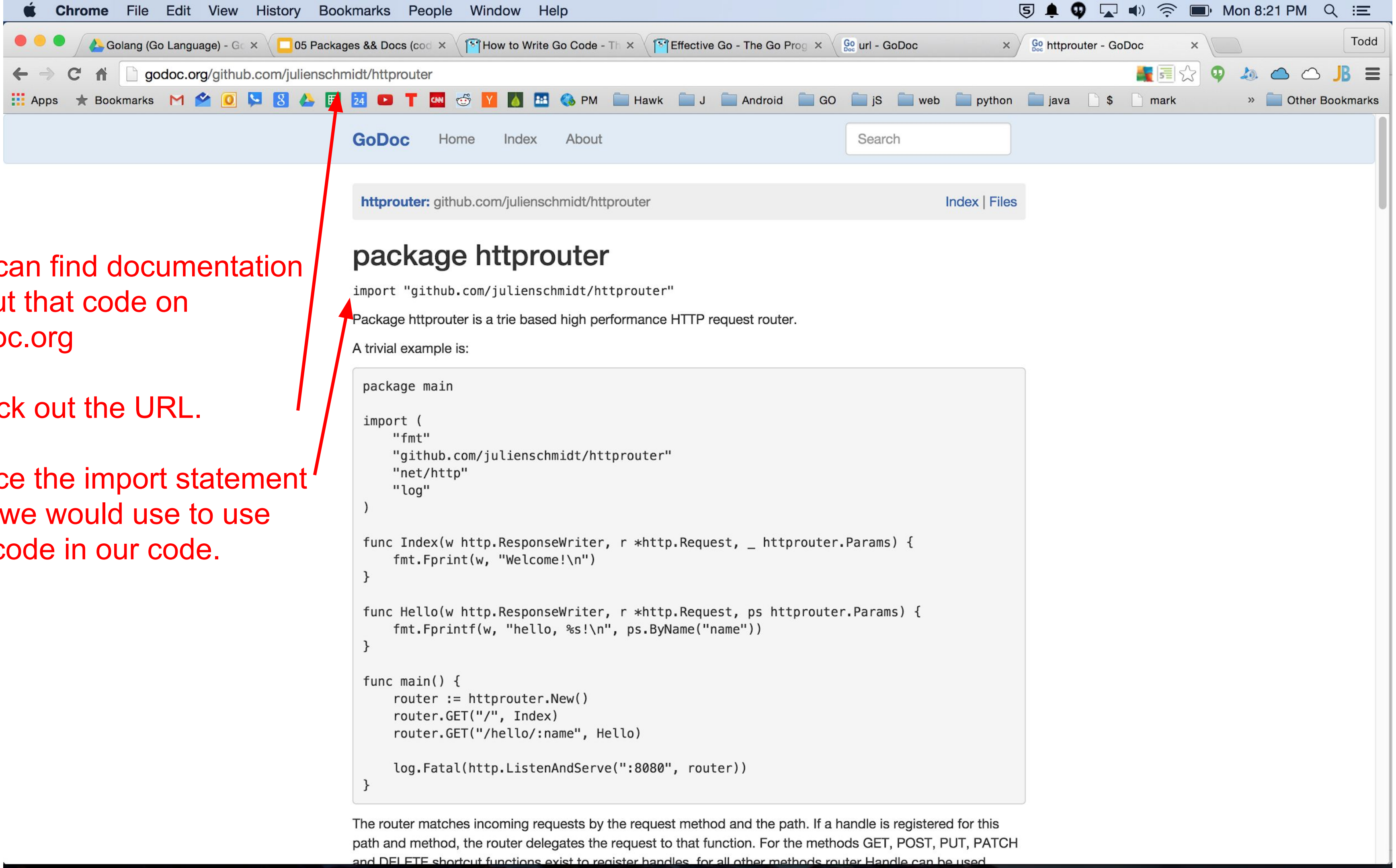


Given this namespace / package name ...

We can find documentation about that code on godoc.org

Check out the URL.

Notice the import statement that we would use to use the code in our code.


Chrome File Edit View History Bookmarks People Window Help
Mon 8:21 PM
Golang (Go Language) - G
05 Packages && Docs (cod
How to Write Go Code - Th
Effective Go - The Go Prog
url - GoDoc
httprouter - GoDoc
Todd
godoc.org/github.com/julienschmidt/httprouter
Apps Bookmarks PM Hawk J Android GO jS web python java $ mark Other Bookmarks
GoDoc Home Index About Search
Index | Files

httprouter: github.com/julienschmidt/httprouter

package httprouter

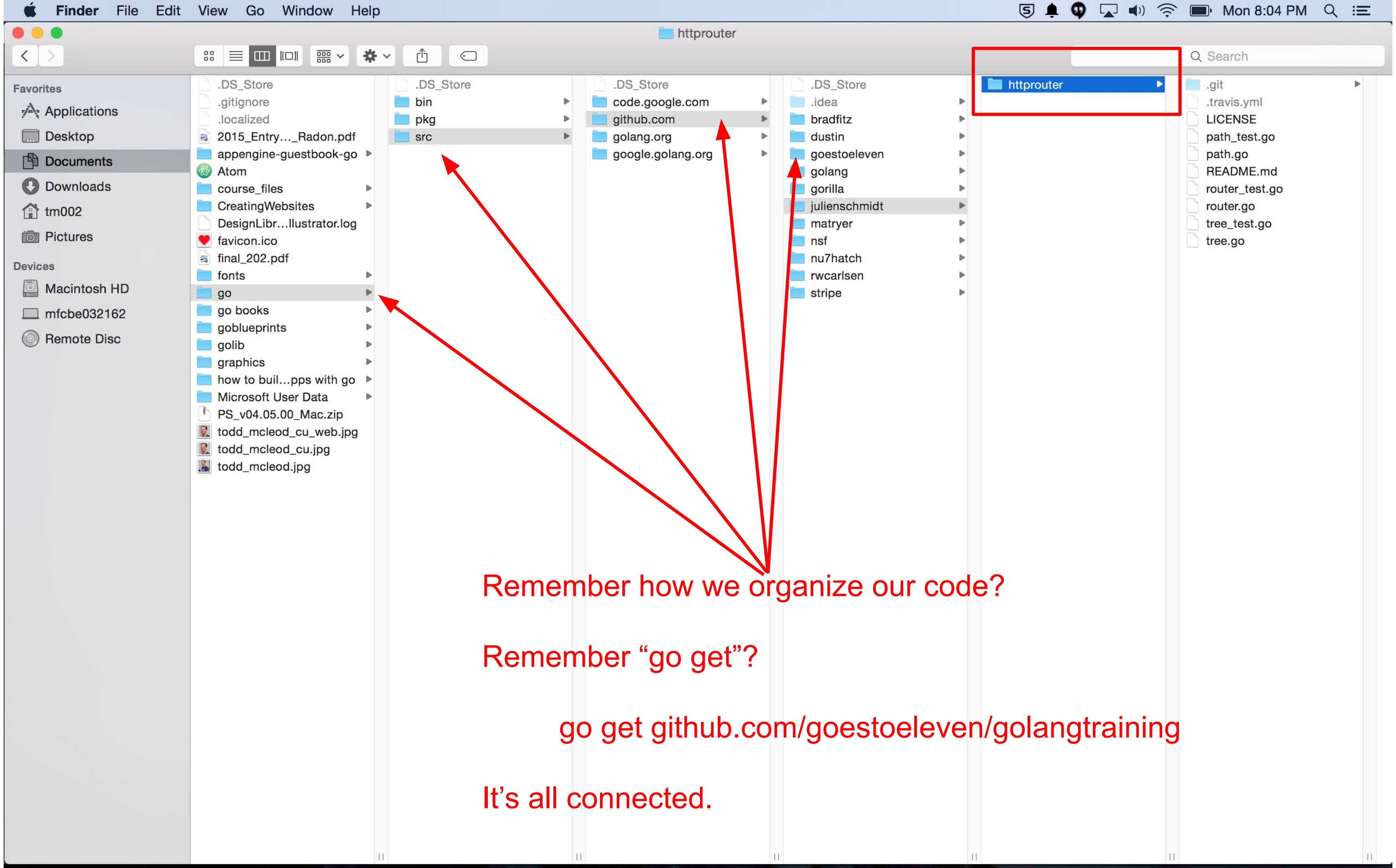
```
import "github.com/julienschmidt/httprouter"
```

Package httprouter is a trie based high performance HTTP request router.

A trivial example is:

```
package main

import (
    "fmt"
    "github.com/julienschmidt/httprouter"
    "net/http"
    "log"
)

func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
    fmt.Fprint(w, "Welcome!\n")
}

func Hello(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
    fmt.Fprintf(w, "hello, %s!\n", ps.ByName("name"))
}

func main() {
    router := httprouter.New()
    router.GET("/", Index)
    router.GET("/hello/:name", Hello)

    log.Fatal(http.ListenAndServe(":8080", router))
}
```

The router matches incoming requests by the request method and the path. If a handle is registered for this path and method, the router delegates the request to that function. For the methods GET, POST, PUT, PATCH and DELETE shortcut functions exist to register handles, for all other methods router.Handle can be used



Remember how we organize our code?

Remember “go get”?

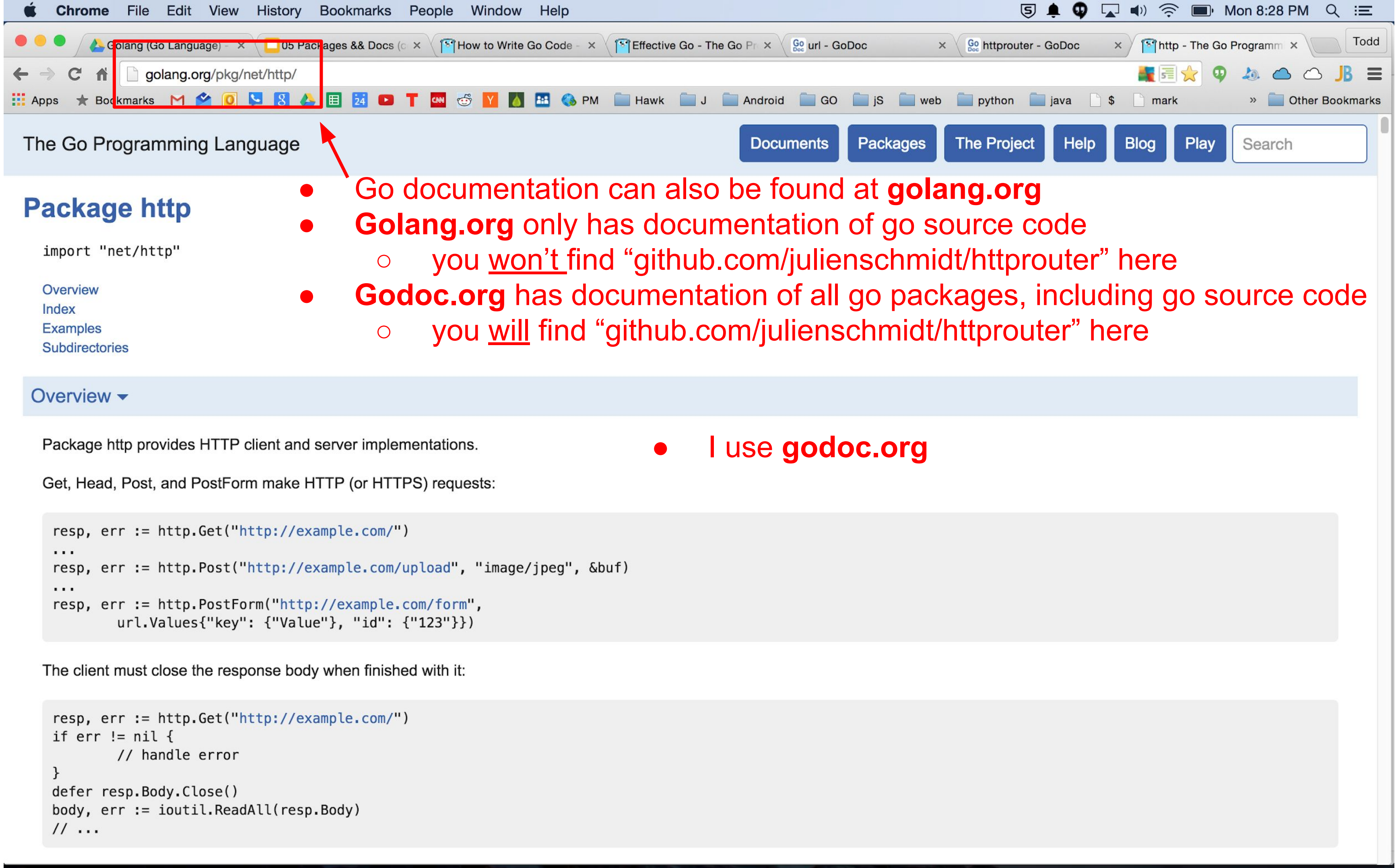
go get github.com/goestoeleven/golangtraining

It’s all connected.

documentation

godoc.org vs golang.org vs. godoc at terminal

I mainly use godoc.org as it includes the packages of others.


Chrome File Edit View History Bookmarks People Window Help
Mon 8:28 PM
golang.org/pkg/net/http/
The Go Programming Language
Documents
Packages
The Project
Help
Blog
Play
Search

- Go documentation can also be found at **golang.org**
- **Golang.org** only has documentation of go source code
  - you won't find "github.com/julienschmidt/httprouter" here
- **Godoc.org** has documentation of all go packages, including go source code
  - you will find "github.com/julienschmidt/httprouter" here

Package http

```
import "net/http"
```

Overview
Index
Examples
Subdirectories

Overview

Package http provides HTTP client and server implementations.

- I use **godoc.org**

Get, Head, Post, and PostForm make HTTP (or HTTPS) requests:

```
resp, err := http.Get("http://example.com/")
...
resp, err := http.Post("http://example.com/upload", "image/jpeg", &buf)
...
resp, err := http.PostForm("http://example.com/form",
	url.Values{"key": {"Value"}, "id": {"123"}})
```

The client must close the response body when finished with it:

```
resp, err := http.Get("http://example.com/")
if err != nil {
	// handle error
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
// ...
```

package naming

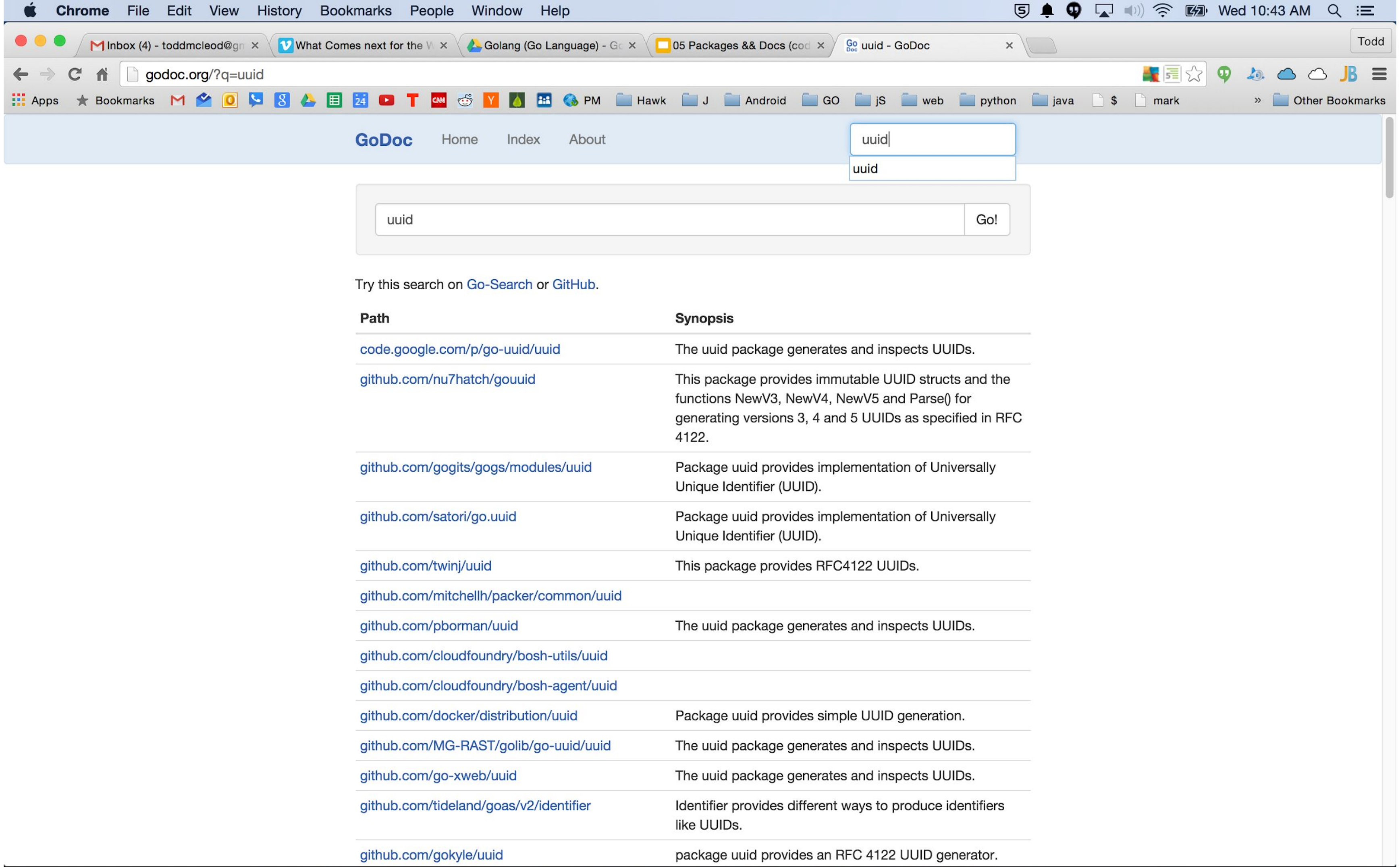
https://golang.org/doc/code.html#PackageNames

package naming

https://golang.org/doc/effective_go.html#names

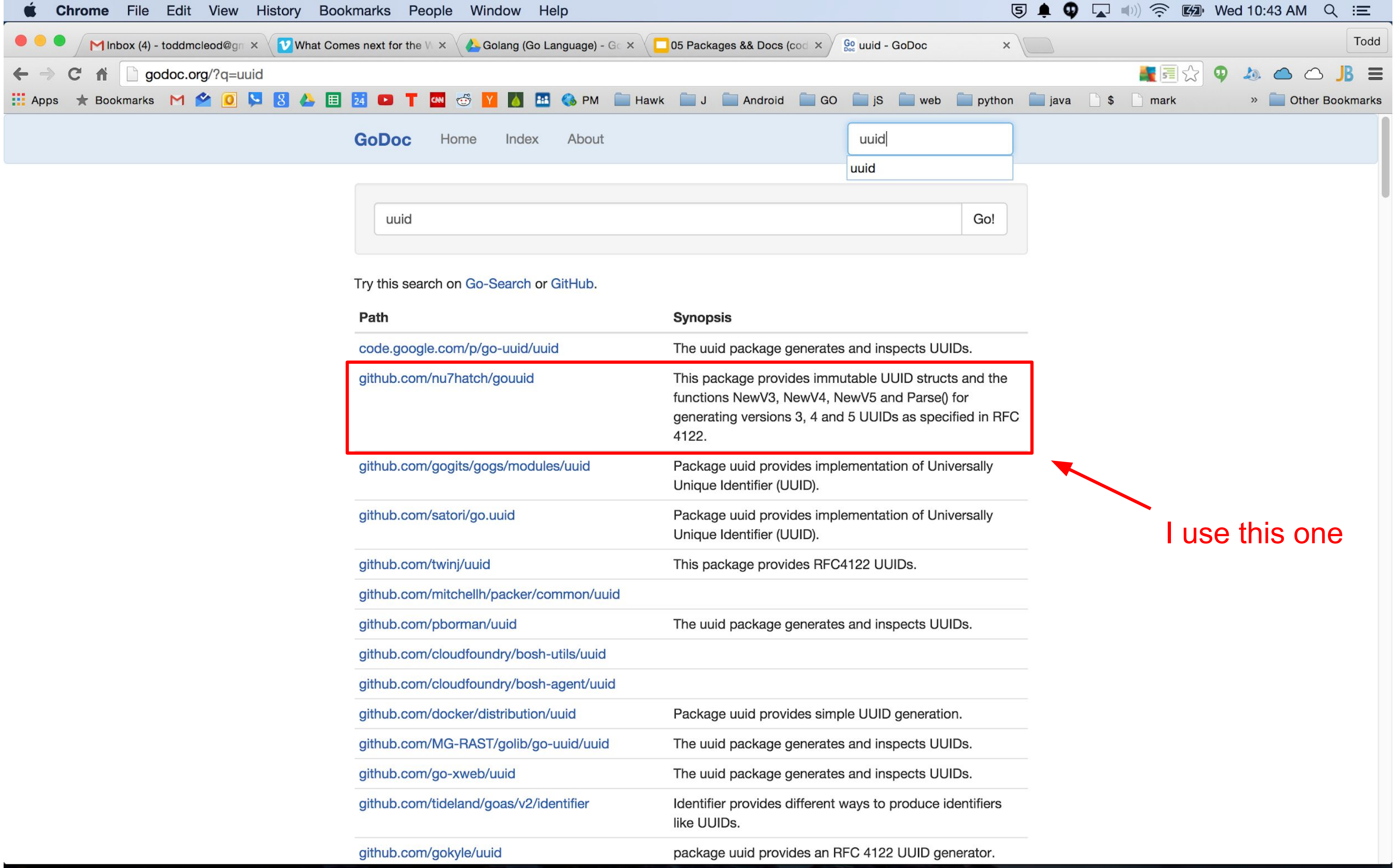
searching for packages / libraries

search godoc.org



Try this search on Go-Search or GitHub.

Path	Synopsis
code.google.com/p/go-uuid/uuid	The uuid package generates and inspects UUIDs.
github.com/nu7hatch/gouuid	This package provides immutable UUID structs and the functions NewV3, NewV4, NewV5 and Parse() for generating versions 3, 4 and 5 UUIDs as specified in RFC 4122.
github.com/gogits/gogs/modules/uuid	Package uuid provides implementation of Universally Unique Identifier (UUID).
github.com/satori/go.uuid	Package uuid provides implementation of Universally Unique Identifier (UUID).
github.com/twinj/uuid	This package provides RFC4122 UUIDs.
github.com/mitchellh/packer/common/uuid	
github.com/pborman/uuid	The uuid package generates and inspects UUIDs.
github.com/cloudfoundry/bosh-utils/uuid	
github.com/cloudfoundry/bosh-agent/uuid	
github.com/docker/distribution/uuid	Package uuid provides simple UUID generation.
github.com/MG-RAST/golib/go-uuid/uuid	The uuid package generates and inspects UUIDs.
github.com/go-xweb/uuid	The uuid package generates and inspects UUIDs.
github.com/tideland/goas/v2/identifier	Identifier provides different ways to produce identifiers like UUIDs.
github.com/gokyle/uuid	package uuid provides an RFC 4122 UUID generator.


Chrome File Edit View History Bookmarks People Window Help
Wed 10:43 AM
Inbox (4) - toddmcleod@gm
What Comes next for the W
Golang (Go Language) - Go
05 Packages && Docs (cod
uuid - GoDoc
godoc.org/?q=uuid
GoDoc
Home
Index
About
uuid
Go!

Try this search on Go-Search or GitHub.

Path	Synopsis
code.google.com/p/go-uuid/uuid	The uuid package generates and inspects UUIDs.
github.com/nu7hatch/gouuid	This package provides immutable UUID structs and the functions NewV3, NewV4, NewV5 and Parse() for generating versions 3, 4 and 5 UUIDs as specified in RFC 4122.
github.com/gogits/gogs/modules/uuid	Package uuid provides implementation of Universally Unique Identifier (UUID).
github.com/satori/go.uuid	Package uuid provides implementation of Universally Unique Identifier (UUID).
github.com/twinj/uuid	This package provides RFC4122 UUIDs.
github.com/mitchellh/packer/common/uuid	
github.com/pborman/uuid	The uuid package generates and inspects UUIDs.
github.com/cloudfoundry/bosh-utils/uuid	
github.com/cloudfoundry/bosh-agent/uuid	
github.com/docker/distribution/uuid	Package uuid provides simple UUID generation.
github.com/MG-RAST/golib/go-uuid/uuid	The uuid package generates and inspects UUIDs.
github.com/go-xweb/uuid	The uuid package generates and inspects UUIDs.
github.com/tideland/goas/v2/identifier	Identifier provides different ways to produce identifiers like UUIDs.
github.com/gokyle/uuid	package uuid provides an RFC 4122 UUID generator.

I use this one

Review

- **packages**
  - packages = folders
- SDK - software development kit
- **namespace**
  - unique namespace of packages: everything after the “src” folder
- documentation
  - **godoc.org**
  - golang.org
  - godoc at terminal
- package names / paths
- searching for packages

Review Review - You’ve learned a lot

- golang is awesome
- SHA1
- **go version**
- **go env**
- **go help**
- environment variables
  - GOPATH
  - GOROOT
- workspace
  - bin
  - pkg
  - src
    - github.com
      - your_user_name
        - your_packages
          - your code
- .bash_profile / .bashrc
- GO IDE’s
  - webstorm
  - atom
- func main()
- packages
- functions vs methods
- parameters vs arguments
- expressions vs statements
- variable, constant, literal
- **go run**
- **go build**
- **go install**
- **go get**
  - go commands
- Go, Github, & Webstorm
- git
  - git log
  - .gitignore
- Packages / Libraries
  - naming
- namespace
- documentation
  - **godoc.org**
  - golang.org
  - godoc at terminal

Review Questions

Namespacing

Define **namespace**.

go commands

List and define the go commands you have learned so far.

I know this is repetitive, but repetition helps us learn.

godoc.org

- Find a package for **gorilla sessions** on godoc.org
    - take a screenshot of this

namespace

- Open up finder (or windows explorer, for those on windows).
- Navigate to your workspace.
- Navigate to the folder you created to store the code you write in this course.
- What is the unique name for this package?
    - for your homework submission
        - write out the unique name for this package and also
        - take a screenshot of finder showing your package