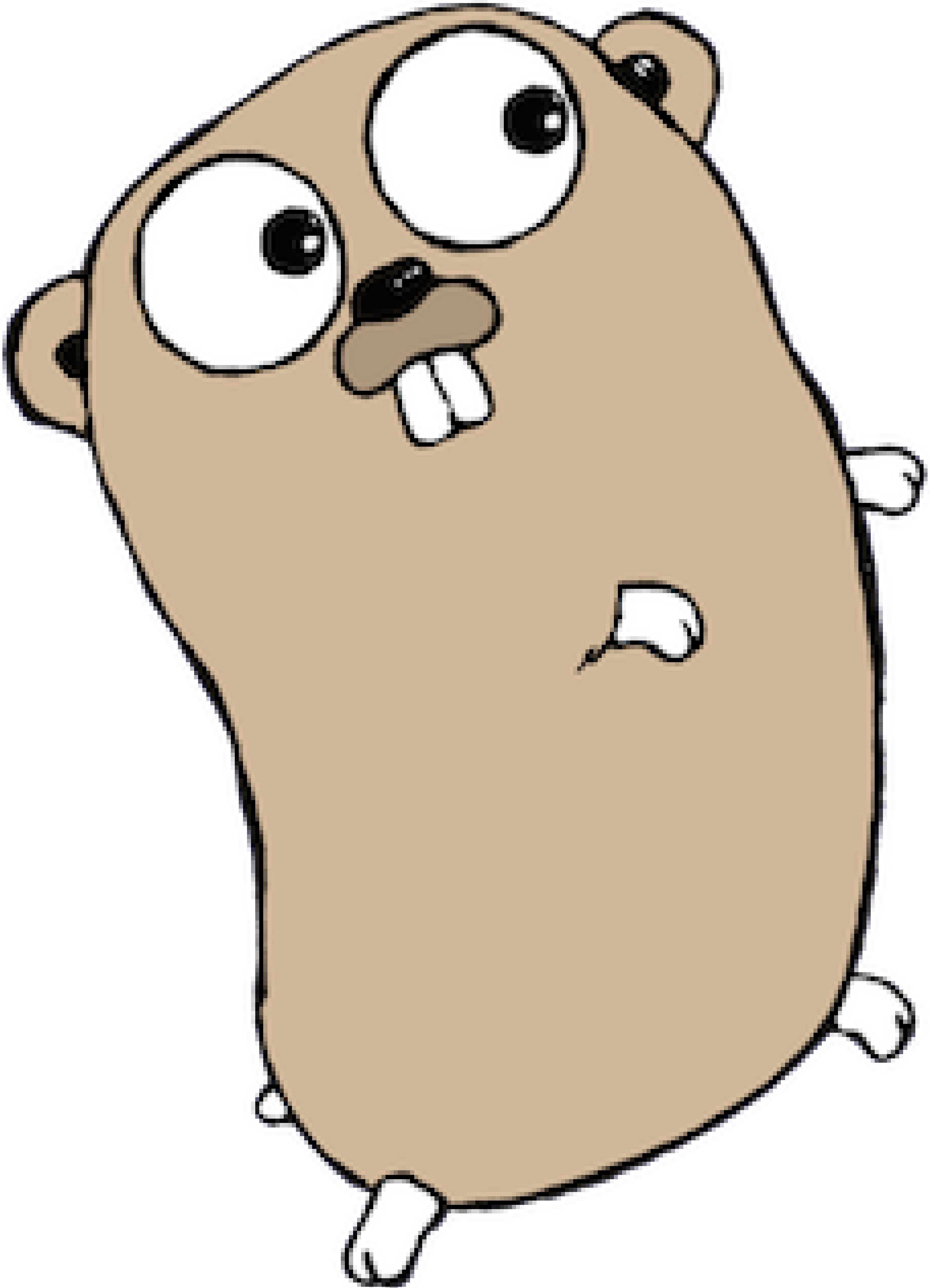


How do I use Sourcegraph with Go?

Satish Talim

This book is for sale at http://leanpub.com/howdoiusesourcegraph

This version was published on 2014-06-14

Tweet This Book!

Please help Satish Talim by spreading the word about this book on Twitter!

The suggested hashtag for this book is #howdoiusesourcegraph.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#howdoiusesourcegraph

Also By Satish Talim

Contents

Preface

Go[1] is an open source programming language that makes it easy to build simple, reliable, and efficient software.

Who is the booklet for?

This short booklet will show Go newbies how one can use Sourcegraph to better write one's Go programs. To try out the programs in this booklet, you should have a working copy of Go 1.2 on your computer.

Acknowledgements

I would like to thank Sourcegraph[2] for permiting me to write this booklet.

Using Code Examples

All of the code in this booklet can be used pretty much anywhere and anyhow you please.

How to Contact Me

I can be reached via e-mail at satish.talim@gmail.com. Please contact me if you have any questions, comments, kudos or criticism on the booklet. Constructive criticism is definitely appreciated; I want this booklet to get better through your feedback.

Thanks

Thanks for downloading and checking out this booklet. As part of the lean publishing philosophy, you'll be able to interact with me as the booklet is completed. I'll be able to change things, reorganize parts, and generally make a better booklet. I hope you enjoy.

[1]http://golang.org/
[2]https://sourcegraph.com/

1 What's Sourcegraph?

Sourcegraph is a code search engine that shows you documentation and real-world usage examples for hundreds of thousands of libraries written in Go and Ruby.

1.1 Getting Started

Sign up[1] with your GitHub account (no private data is requested). Signing in is optional, but it helps Sourcegraph find all of your open-source code and attribute it to you.

1.2 How Do I Use It?

We shall build a small Go program and use Sourcegraph along the way. This simple application: given a GitHub username, it fetches that person's company and full name.

1.3 Assumptions

I am assuming that you have:

- downloaded and installed Go.
- set the environment variable `GOROOT`.
- set up the workspace and environment variable `GOPATH`.
- updated the system environment variable `PATH` to include your workspace `bin` subdirectory.
- setup the package path. I have it set to `c:\go_projects\go\src\github.com\SatishTalim`

Open a command window, make a new folder and cd to it as follows:

```
$ mkdir $GOPATH/src/github.com/SatishTalim/sourcegraph
$ cd $GOPATH/src/github.com/SatishTalim/sourcegraph
```

If you are new to Go, you can download my free eBook[2] that covers all of the above.

1.4 sourcegraph.go - Outline 1

Let's get started. I have a very basic outline of the code `sourcegraph.go`.

[1]https://sourcegraph.com/join
[2]https://leanpub.com/buildingapackageingo

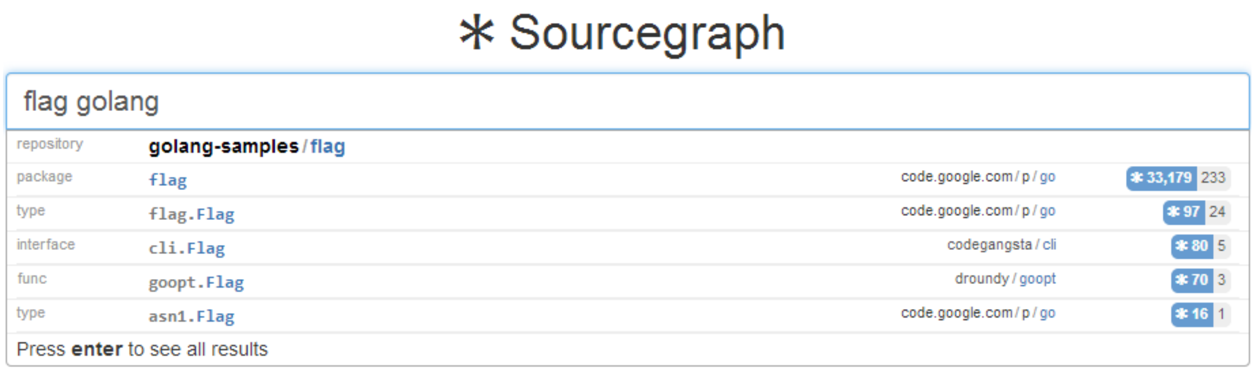
Program sourcegraph.go

```go
package main

func main() {
        // TODO
}
```

First, I want to make a Http request to the GitHub API. Next I would like to parse the JSON response and determine that person's name and company.

I want to take some user input using the **`flag`** package. However, I haven't used this package much and bad at remembering how to exactly use it. I would definitely like to know how. So, let us look it up on sourcegraph[3].

Let's type **`flag golang`** as shown below:



✻ Sourcegraph

flag golang

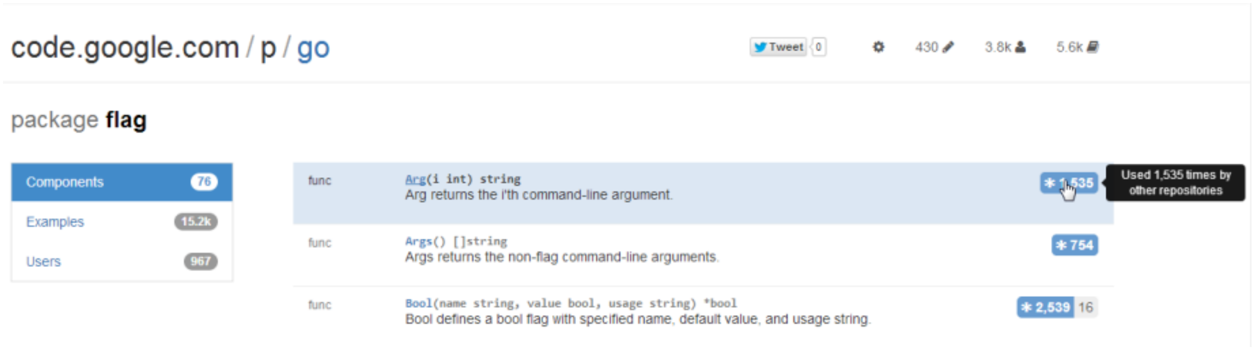
repository	golang-samples / flag		
package	flag	code.google.com / p / go	✻ 33,179 233
type	flag.Flag	code.google.com / p / go	✻ 97 24
interface	cli.Flag	codegangsta / cli	✻ 80 5
func	goopt.Flag	droundy / goopt	✻ 70 3
type	asn1.Flag	code.google.com / p / go	✻ 16 1

Press **enter** to see all results

Sourcegraph

If you see **`package flag`** there are over 33,000 examples of usage of **`flag`** on sourcegraph. That's good. Let's click on that.

In the image below, you can see the functions and other definitions in the package **`flag`**. On the right you can see how many times it has been used by other people.



code.google.com / p / go

Tweet 0 430 3.8k 5.6k

package **flag**

Components 76
Examples 15.2k
Users 967

func `Arg(i int) string`
Arg returns the i'th command-line argument. ✻ 1,535 — Used 1,535 times by other repositories

func `Args() []string`
Args returns the non-flag command-line arguments. ✻ 754

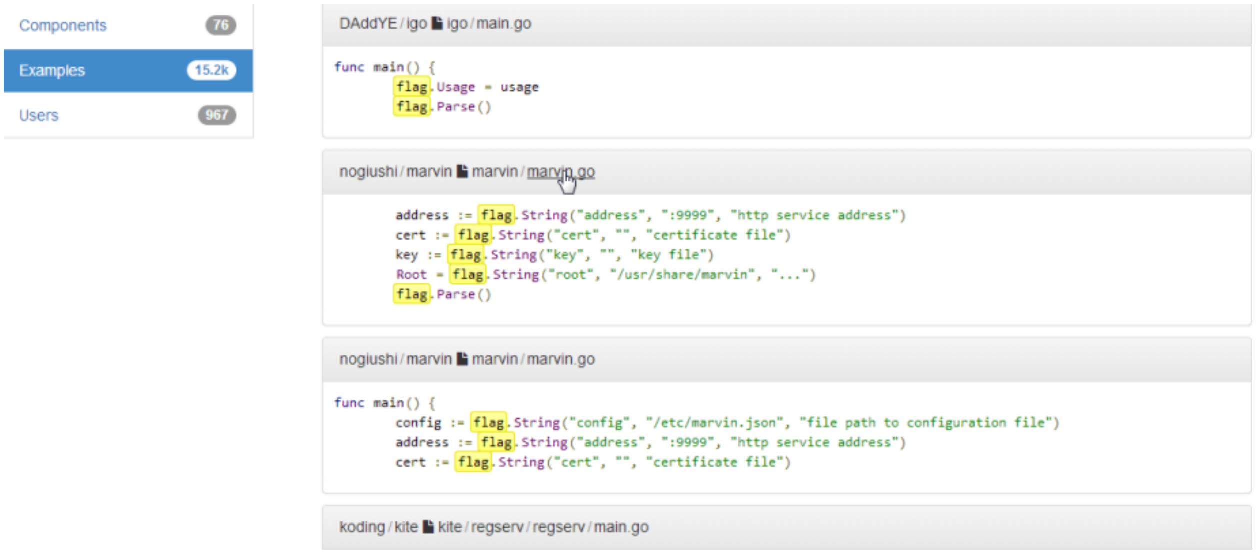
func `Bool(name string, value bool, usage string) *bool`
Bool defines a bool flag with specified name, default value, and usage string. ✻ 2,539 16

Sourcegraph

[3] https://sourcegraph.com/

Click on the Examples link on the left and see some quick examples of how it is used.

Scroll down and see which example is similar to the one you want to write. I think **`marvin.go`** seems to be what I want.



DAddYE / igo igo / main.go

```
func main() {
        flag.Usage = usage
        flag.Parse()
```

nogiushi / marvin marvin / marvin.go

```
        address := flag.String("address", ":9999", "http service address")
        cert := flag.String("cert", "", "certificate file")
        key := flag.String("key", "", "key file")
        Root = flag.String("root", "/usr/share/marvin", "...")
        flag.Parse()
```

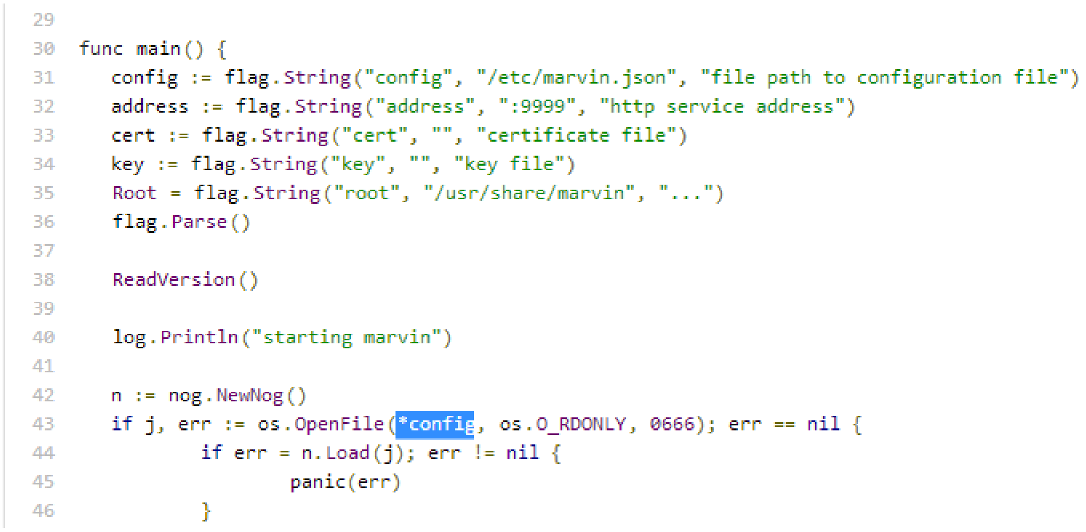
nogiushi / marvin marvin / marvin.go

```
func main() {
        config := flag.String("config", "/etc/marvin.json", "file path to configuration file")
        address := flag.String("address", ":9999", "http service address")
        cert := flag.String("cert", "", "certificate file")
```

koding / kite kite / regserv / regserv / main.go

Sourcegraph

Click on **`marvin.go`** to load the full example as seen in the image below.



```

func main() {
   config := flag.String("config", "/etc/marvin.json", "file path to configuration file")
   address := flag.String("address", ":9999", "http service address")
   cert := flag.String("cert", "", "certificate file")
   key := flag.String("key", "", "key file")
   Root = flag.String("root", "/usr/share/marvin", "...")
   flag.Parse()

   ReadVersion()

   log.Println("starting marvin")

   n := nog.NewNog()
   if j, err := os.OpenFile(*config, os.O_RDONLY, 0666); err == nil {
           if err = n.Load(j); err != nil {
                   panic(err)
           }
```

Sourcegraph

I think I will use **`flag.String`** and **`flag.Parse()`** in my program **`sourcegraph.go`**.

1.5 sourcegraph.go - Outline 2

Let's type in the program as follows:

Program sourcegraph.go

```
package main

import (
        "flag"
        "log"
)

func main() {
        login := flag.String("login", "", "GitHub login of user")
        flag.Parse()

        if login == "" {
                log.Fatal("must specify login")
        }

        log.Println("Looking up GitHub user: ", login)
}
```

In the program above, we are using the **`flag`** package to read in the command line arguments.

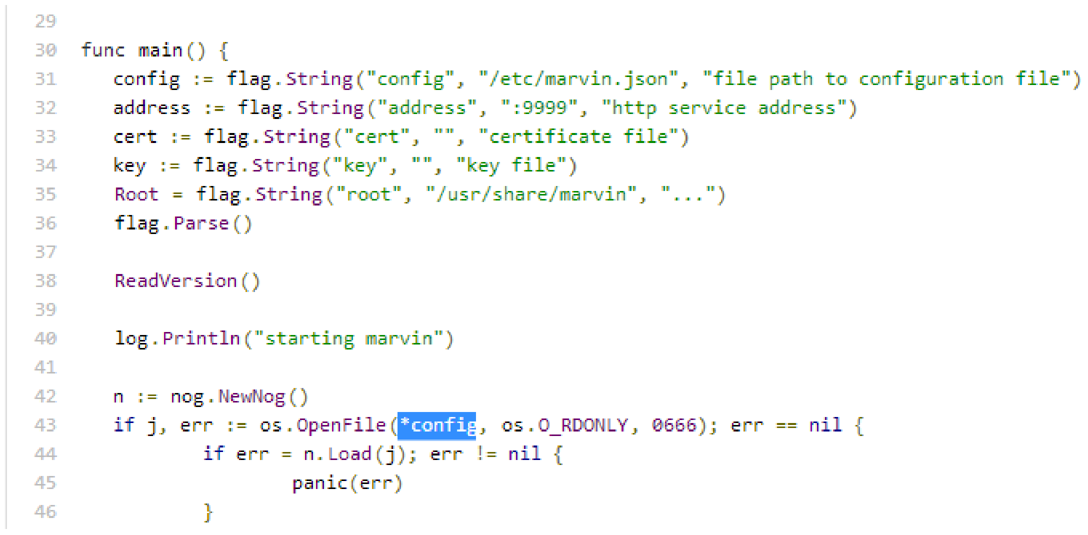
Next, in the same folder where the program is located, type:

```
$ go run sourcegraph.go
```

Oh! We get an error that says:

```
login == "" (mismatched types *string and string)
```

If you re-check the image below, you will find that we need to de-reference the variable (see **`*config`**):



```

func main() {
	config := flag.String("config", "/etc/marvin.json", "file path to configuration file")
	address := flag.String("address", ":9999", "http service address")
	cert := flag.String("cert", "", "certificate file")
	key := flag.String("key", "", "key file")
	Root = flag.String("root", "/usr/share/marvin", "...")
	flag.Parse()

	ReadVersion()

	log.Println("starting marvin")

	n := nog.NewNog()
	if j, err := os.OpenFile(*config, os.O_RDONLY, 0666); err == nil {
		if err = n.Load(j); err != nil {
			panic(err)
		}
```

Sourcegraph

1.6 sourcegraph.go - Outline 3

Our modified program is:

Program sourcegraph.go

```go
package main

import (
        "flag"
        "log"
)

func main() {
        login := flag.String("login", "", "GitHub login of user")
        flag.Parse()

        if *login == "" {
                log.Fatal("must specify login")
        }

        log.Println("Looking up GitHub user: ", *login)
}
```

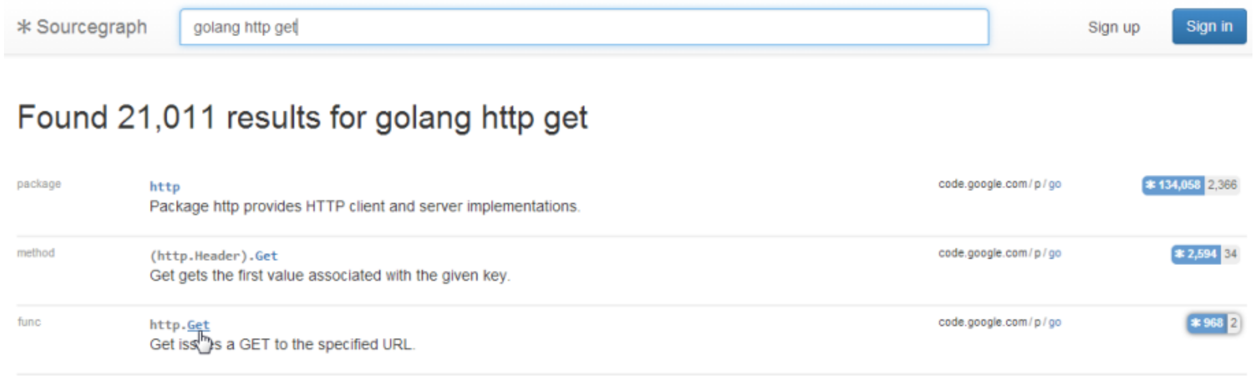
Let us re-run our program:

```
$ go run sourcegraph.go
2014/06/03 09:49:50 must specify login
```

`must specify login` great! Okay, let us do that with my login **`SatishTalim`**:

```
$ go run sourcegraph.go -login=SatishTalim
2014/06/03 09:45:30 Looking up GitHub user:  SatishTalim
```

Cool! It's working. Now, let us write the code that fetches from the GitHub API. Let's go back to Sourcegraph and see how to do a **`get`**.

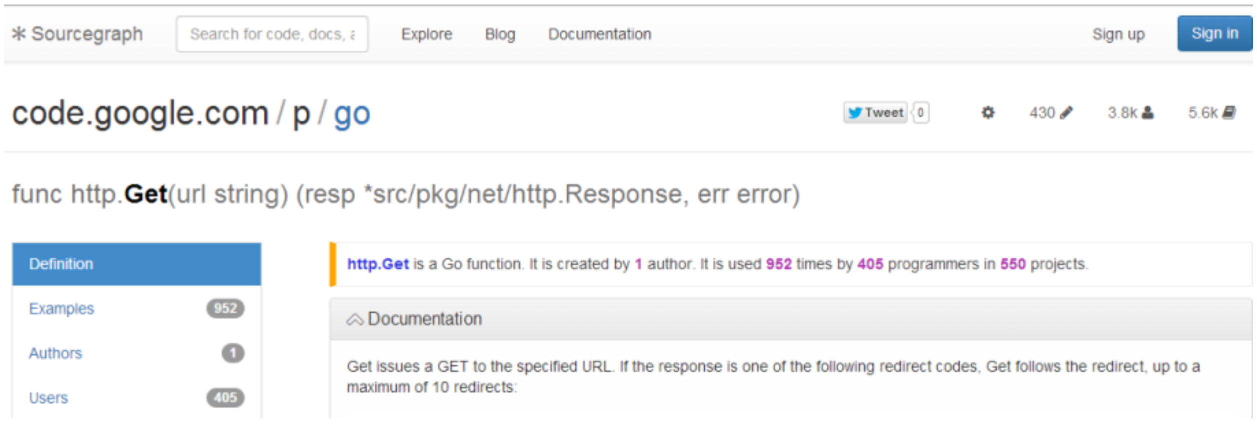

✻ Sourcegraph | golang http get | Sign up | Sign in

Found 21,011 results for golang http get

package	`http` Package http provides HTTP client and server implementations.	code.google.com / p / go	✻ 134,058 2,366
method	`(http.Header).Get` Get gets the first value associated with the given key.	code.google.com / p / go	✻ 2,594 34
func	`http.Get` Get issues a GET to the specified URL.	code.google.com / p / go	✻ 968 2

Sourcegraph

Let’s click on **`http.Get`**. We should see as shown in the image below:


✻ Sourcegraph | Search for code, docs, a | Explore | Blog | Documentation | Sign up | Sign in

code.google.com / p / go — Tweet 0 — 430 — 3.8k — 5.6k

func http.**Get**(url string) (resp *src/pkg/net/http.Response, err error)

Definition
Examples 952
Authors 1
Users 405

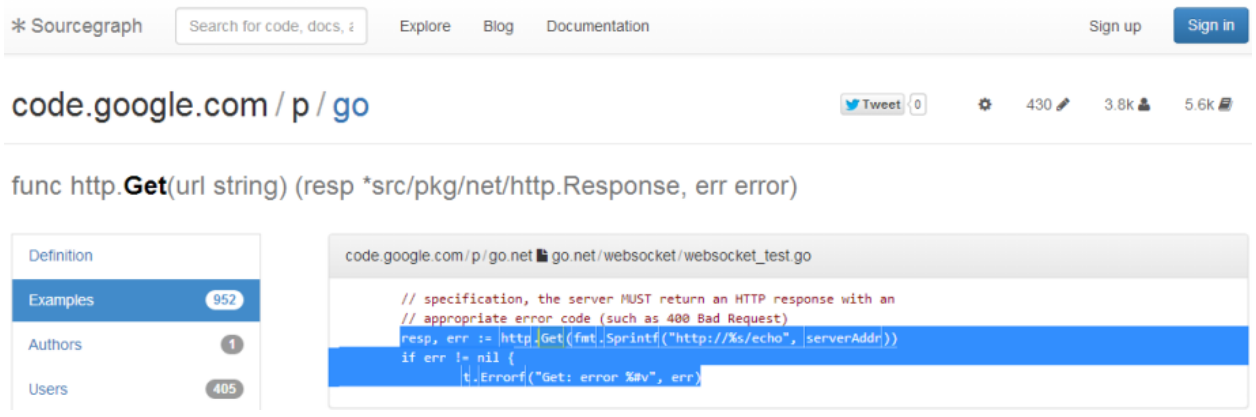
http.Get is a Go function. It is created by **1** author. It is used **952** times by **405** programmers in **550** projects.

Documentation

Get issues a GET to the specified URL. If the response is one of the following redirect codes, Get follows the redirect, up to a maximum of 10 redirects:

Sourcegraph

In the image, we can see the documentation, the **`http.Get`** being used by 405 programmers and so on. Let’s click on Examples on the left.


✻ Sourcegraph | Search for code, docs, a | Explore | Blog | Documentation | Sign up | Sign in

code.google.com / p / go — Tweet 0 — 430 — 3.8k — 5.6k

func http.**Get**(url string) (resp *src/pkg/net/http.Response, err error)

Definition
Examples 952
Authors 1
Users 405

code.google.com / p / go.net / go.net / websocket / websocket_test.go

```
        // specification, the server MUST return an HTTP response with an
        // appropriate error code (such as 400 Bad Request)
        resp, err := http.Get(fmt.Sprintf("http://%s/echo", serverAddr))
        if err != nil {
                t.Errorf("Get: error %#v", err)
```

Sourcegraph

I think the highlighted text seems like a simple example, which I will copy/paste into my program.

1.7 sourcegraph.go - Outline 4

Having already looked up the GitHub API documentation[4], I have edited the copied code slightly as in:

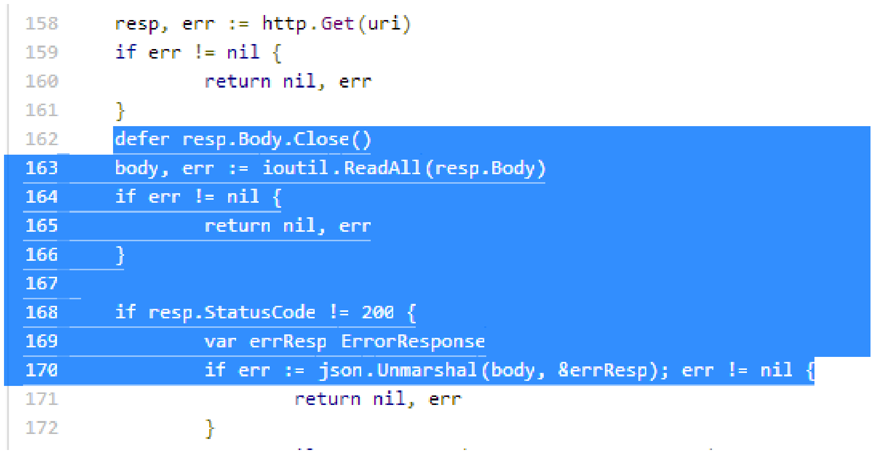
Program sourcegraph.go

```go
package main

import (
	"flag"
	"fmt"
	"log"
	"net/http"
)

func main() {
	login := flag.String("login", "", "GitHub login of user")
	flag.Parse()

	if *login == "" {
		log.Fatal("must specify login")
	}

	log.Println("Looking up GitHub user: ", *login)

	resp, err := http.Get(fmt.Sprintf("http://api.github.com/users/%s", *login))
	if err != nil {
		log.Fatal(err)
	}
}
```

Note that I have added two libraries namely **`fmt`** and **`net/http`**. In this program we are able to get the response **`resp`** but how do we get it's contents?

We had copied/pasted the code from the **`websocket_test.go`** program. Let's go back to its full program listing and check if we find what to do with **`resp`**. You will soon realize that there's nothing related to **`resp`** there. Let us go back to the other examples listed where **`websocket_test.go`** is. Scrolling down, let us check **`docker/utils/utils.go`**. Oh! Nothing here. On the next page I plan to look at **`hipchat/hipchat.go`** as shown below:

[4] https://developer.github.com/v3/



```
resp, err := http.Get(uri)
if err != nil {
        return nil, err
}
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
if err != nil {
        return nil, err
}

if resp.StatusCode != 200 {
        var errResp ErrorResponse
        if err := json.Unmarshal(body, &errResp); err != nil {
                return nil, err
        }
```

Sourcegraph

I observe that I can extract the **`resp.Body`** out of **`resp`** and then use **`json.Unmarshal`** to extract the contents namely the person’s name and company. Observe that **`json.Unmarshal`** extracts the information into a variable. We need to provide a variable where the JSON package can put the decoded data. This **`map[string]interface{}`** will hold a map of strings to arbitrary data types.

1.8 sourcegraph.go - Outline 5

Let us copy/paste the relevant code from `hipchat/hipchat.go` into our program:

Program sourcegraph.go

```go
package main

import (
	"encoding/json"
	"flag"
	"fmt"
	"io/ioutil"
	"log"
	"net/http"
)

func main() {
	login := flag.String("login", "", "GitHub login of user")
	flag.Parse()

	if *login == "" {
		log.Fatal("must specify login")
	}

	log.Println("Looking up GitHub user: ", *login)

	resp, err := http.Get(fmt.Sprintf("https://api.github.com/users/%s", *login))
	if err != nil {
		log.Fatal(err)
	}

	defer resp.Body.Close()
	body, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		log.Fatal(err)
	}

	var userdata map[string]interface{}
	err = json.Unmarshal(body, &userdata)
	if err != nil {
		log.Fatal(err)
	}
	log.Println(userdata)
}
```

Remember to import the package `io/ioutil` and the last statement `log.Println(userdata)` just prints the data.

Let us run our program:

```
$ go run sourcegraph.go -login=SatishTalim
2014/06/04 09:57:34 Looking up GitHub user:  SatishTalim
2014/06/04 09:57:35 map[organizations_url:https://api.github.com/users/SatishTal
im/orgs type:User hireable:false bio:Founder RubyLearning public_gists:40 avatar
_url:https://avatars.githubusercontent.com/u/1052069? followers_url:https://api.
github.com/users/SatishTalim/followers location:India public_repos:26 followers:
67 gists_url:https://api.github.com/users/SatishTalim/gists{/gist_id} created_at
:2011-09-15T03:47:18Z login:SatishTalim url:https://api.github.com/users/SatishT
alim starred_url:https://api.github.com/users/SatishTalim/starred{/owner}{/repo}
 repos_url:https://api.github.com/users/SatishTalim/repos name:Satish Talim comp
any:RubyLearning subscriptions_url:https://api.github.com/users/SatishTalim/subs
criptions email:satish@rubylearning.org id:1.052069e+06 html_url:https://github.
com/SatishTalim following_url:https://api.github.com/users/SatishTalim/following
{/other_user} following:7 gravatar_id:e78a9914ef2e253f5af78768600f9193 blog:http
://rubylearning.org/ events_url:https://api.github.com/users/SatishTalim/events{
/privacy} received_events_url:https://api.github.com/users/SatishTalim/received_
events site_admin:false updated_at:2014-06-04T04:15:11Z]
```

You can see the API response above, but we want to show some specific pieces of information. We can see the `name` and `company` in that data.

1.9 sourcegraph.go - Final program

Instead of printing **`userdata`** we will print only the required data as shown below:

Program sourcegraph.go

```go
package main

import (
	"encoding/json"
	"flag"
	"fmt"
	"io/ioutil"
	"log"
	"net/http"
)

func main() {
	login := flag.String("login", "", "GitHub login of user")
	flag.Parse()

	if *login == "" {
		log.Fatal("must specify login")
	}

	log.Println("Looking up GitHub user: ", *login)

	resp, err := http.Get(fmt.Sprintf("https://api.github.com/users/%s", *login))
	if err != nil {
		log.Fatal(err)
	}

	defer resp.Body.Close()
	body, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		log.Fatal(err)
	}

	var userdata map[string]interface{}
	err = json.Unmarshal(body, &userdata)
	if err != nil {
		log.Fatal(err)
	}
	log.Println("User's full name: ", userdata["name"])
	log.Println("User's company: ", userdata["company"])
}
```

Re-run the program:

```
$ go run sourcegraph.go -login=SatishTalim
2014/06/04 10:08:56 Looking up GitHub user:  SatishTalim
2014/06/04 10:08:57 User's full name:  Satish Talim
2014/06/04 10:08:57 User's company:  RubyLearning
```

That's my data and it's correct. Let us try another user:

```
$ go run sourcegraph.go -login=beyang
2014/06/04 10:11:43 Looking up GitHub user:  beyang
2014/06/04 10:11:44 User's full name:  Beyang Liu
2014/06/04 10:11:44 User's company:  Sourcegraph
```

That's it! So searching on Sourcegraph you can quickly see how other programmers are doing similar work that we are doing. We thus save a lot of time by being able to look at code and see how things are actually used, instead of having to read thro' docs. Documentation is great and Sourcegraph is a great way to find docs quickly but sometimes an example is worth thousands of lines of documentation.

So try Sourcegraph out and if you have any feedback for them, write to .

Sourcegraph is working really hard to make it the best tool for open source programmers.