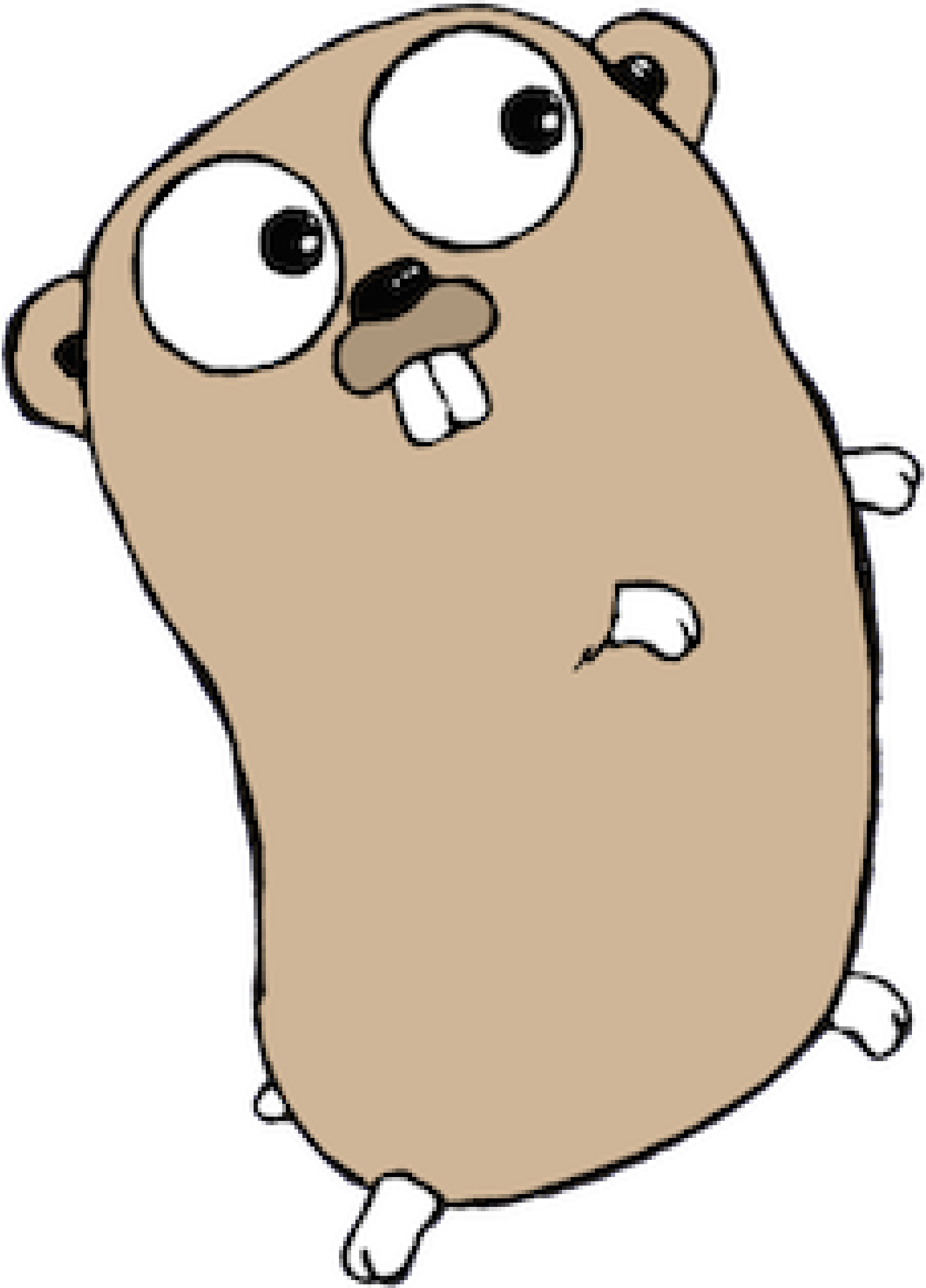

Building a package in Go

Satish Talim

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

This version was published on 2014-12-28

Tweet This Book!

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

The suggested hashtag for this book is #bapigo.

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=#bapigo

Also By Satish Talim

How Do I Write And Deploy Simple Web Apps With Go?

How do I use Sourcegraph with Go?

How do I use Sourcegraph with Ruby?

Contents

Preface

Who is the book for?

I am an experienced programmer but new to the Go programming language. The best way to learn a new programming language is to build something. I decided to build a *barebones* package called `redditnews` (inspired from https://github.com/nf/reddit[1]) and at the same time document what all I did, so that other "baby gophers" (newbies) could learn and benefit.

What will you learn?

In the end, you will understand and know the *mechanics* of creating a package in Go.

Acknowledgements

- I would like to thank the Go community for their help in making this eBook far better than I could have done alone.
- The Gopher character is based on the Go mascot designed by Rene French[2] and copyrighted under the Creative Commons Attribution 3.0 license.

Using Code Examples

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

Getting the Code

You can get a `.zip` or `.tar` archive of the code by going to GitHub Repo[3] and clicking on the "Download ZIP" button.

How to Contact Me

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

---

[1] https://github.com/nf/reddit
[2] http://reneefrench.blogspot.com/
[3] https://github.com/SatishTalim/redditnews

Thanks

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

1. Project - redditnews for Baby Gophers

1.1 Our Project

I am keen to be abreast with what's happening in the **Golang** world. To that end, we will write a command-line program (`redditnews.go`) that fetches and displays the latest headlines from the golang page on Reddit.

The program will:

- make an HTTP request to the Reddit API.
- decode the JSON response into a Go data structure, and
- display each link's author, score, URL and title.

We will then be building a bare-bones News Reader package (`redditnews`) that gives us the latest news and headlines from the Golang Sub-Reddit, using Reddit's API.

Once the project is ready, we will use GMail to send an email which will contain the information fetched by the package `redditnews`.

1.2 What will you learn?

You will learn and understand the *mechanics of creating a package in Go.*

2. Getting Started

As a “baby gopher” I plan to:

- work in iterations (you can think of an iteration like a draft when you’re writing a paper).
- document each and every step as we build this package, and
- use the Go programming language to build this package.

2.1 Downloading Go

Visit the Go project’s downloads page[1] and select the binary distribution that matches your operating system and processor architecture.

2.2 Learning Go

I learnt Go using the following resources:

- Main Golang Site[2]
- A Tour of Go[3]
- Go by Example[4]
- How to Write Go Code[5]
- Effective Go[6]
- Go in Action[7] book
- Error handling and Go[8]
- JSON decoding in Go[9]
- Interfaces: the awesome sauce of Go[10]
- Go Code Review Comments[11]
- Go: Best Practices for Production Environments[12]

[1] https://code.google.com/p/go/wiki/Downloads
[2] http://golang.org/
[3] http://tour.golang.org/#1
[4] https://gobyexample.com/
[5] http://golang.org/doc/code.html
[6] http://golang.org/doc/effective_go.html
[7] http://www.manning.com/ketelsen/
[8] http://blog.golang.org/error-handling-and-go
[9] http://attilaolah.eu/2013/11/29/json-decoding-in-go/
[10] https://go-book.appspot.com/interfaces.html
[11] https://code.google.com/p/go-wiki/wiki/CodeReviewComments
[12] http://peter.bourgon.org/go-in-production/

2.3 Install the Go tools

The Go binary distributions assume they will be installed in **`/usr/local/go`** (or **`c:\go`** under Windows), but it is possible to install them in a different location. If you do this, you will need to set the **`GOROOT`** environment variable to that directory when using the Go tools.

For example, if you installed Go to your home directory you should add the following commands to **`$HOME/.profile`**:

```
export GOROOT=$HOME/go
export PATH=$PATH:$GOROOT/bin
```

Under Windows, you may set environment variables through the "Environment Variables" button on the "Advanced" tab of the "System" control panel. Some versions of Windows provide this control panel through the "Advanced System Settings" option inside the "System" control panel.

The Go-environment works with a small number of OS environment variables. They are *not* required for building the Go-environment, but by setting them and thus overriding the defaults the Go compilation environment can be customized. In my Windows environment, I have set the following:

```
GOROOT  `C:\go`
GOOS    windows
GOARCH  386
```

In the same dialog-window: Edit the **`PATH`** variable as follows: **`C:\go\bin; ...rest of PATH...`**

2.4 Editor for Go

Download and install and use any plain text editor of your choice to write your Go code.

2.5 Formatting code

Gofmt[13] is a tool that automatically formats Go source code.

Instead of arguing about where curly braces should go, or whether to use tabs or spaces when you indent, the tool makes these decisions by applying a pre-determined style to Go source code.

Most seasoned Go developers configure their development environment to perform a **`go fmt`** on save or before committing to a code repository. In the **`misc`** directory in your Go installation there are plugins and instructions for how to do this for many common editors.

To format your code on the command line, you can use the the **`go fmt`** command:

[13] http://golang.org/cmd/gofmt/

```
$ go fmt path/to/your/package
```

2.6 Test your installation

Check that Go is installed correctly by building a simple program, as follows.

Create a file named **hello.go** in some folder (for now) and put the following program in it:

Program hello.go

```
package main

import "fmt"

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

Note: You can run the above program in the Go playgorund[14] too.

Then from the folder where you have saved the file **hello.go**, run it with the **go** tool:

```
$ go run hello.go
hello, world
```

If you see the "hello, world" message then your Go installation is working.

2.7 Use GitHub

One of the best places to share your code with friends, co-workers, classmates, and complete strangers is GitHub[15].

2.7.1 Create an account

On the main screen that shows up, enter your **username**, **your email** and **create a password**. Next, click on **Sign up for GitHub**. On the next screen, The 'Free' plan is automatically chosen for you. Just click on the **Finish sign up** button. You will receive an email asking you to 'Confirm your email'. Please do that.

[14] http://play.golang.org/p/-OL2ksKYLO
[15] https://github.com/

2.8 Set up Git

Git is a free and open source *distributed version control system* designed to handle everything from small to very large projects with speed and efficiency.

If you are new to Git, then learn Git in 15 minutes[16].

Next, Set Up Git[17] on your computer as per the guide.

2.9 Go Code Organization

We will create a simple Go package with the help of the go tool[18], the standard way to fetch, build, and install Go packages and commands.

The `go` tool requires you to organize your code in a specific way.

2.9.1 Workspaces

Go code must be kept inside a *workspace*. A workspace is a directory hierarchy with three directories at its root:

- `src` contains Go source files organized into packages (one package per directory),
- `pkg` contains package objects, and
- `bin` contains executable commands.

The `go` tool builds source packages and installs the resulting binaries to the `pkg` and `bin` directories.

The `src` subdirectory typically contains version control repositories (such as for Git) that track the development of one or more source packages.

2.9.2 The GOPATH environment variable

The `GOPATH` environment variable specifies the location of your workspace.

To get started, create a workspace directory and set `GOPATH` accordingly. Your workspace can be located wherever you like. Note that this must not be the same path as your Go installation.

On my Windows computer, I have set `GOPATH=C:\go_projects\go`. Next I update my system environment variable `PATH` to include my workspace `bin` subdirectory i.e. `PATH=%PATH%;%GOPATH%\bin;`

[16] https://try.github.io/levels/1/challenges/1
[17] https://help.github.com/articles/set-up-git
[18] http://golang.org/cmd/go/

My workspace folder structure

```
C:\go_projects
\---go
    +---bin
    +---pkg
    \---src
```

2.9.3 Package paths

The packages from the standard library are given short paths such as **`fmt`** and **`net/http`**. For your own packages, you must choose a base path that is unlikely to collide with future additions to the standard library or other external libraries. You don't need to worry about collisions. You can always *alias* the package name in the import.

If you have a GitHub account at **`github.com/user`**, that should be your base path. We will use github.com/user as our base path. Create a directory inside your workspace in which to keep source code. I have created the folder **`C:\go_projects\go\src\github.com\SatishTalim`**.

Tip

Replace **`SatishTalim`** with your own username.

3. redditnews.go (First Iteration)

This is the first draft of my program.

In your browser, open the site `http://reddit.com/r/golang.json` the browser output is a huge blob of JSON that we receive from the Golang Subreddit. This may be difficult to look at in the browser, unless you have the JSONView plugin installed. These extensions are available for Firefox[1] and Chrome[2]. With the extension installed, here's a partial view of the JSON:

JSON

Now let's write the first draft of our program.

Make a new folder and cd to it as follows:

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

In this folder, create a file named **`redditnews.go`**, open it in your favorite editor, and add the following lines:

[1] https://addons.mozilla.org/en-us/firefox/addon/jsonview/
[2] https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc

Program redditnews.go

```
package main

import (
        "io"
        "log"
        "net/http"
        "os"
)

func main() {
        resp, err := http.Get("http://reddit.com/r/golang.json")
        if err != nil {
                log.Fatal(err)
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                log.Fatal(resp.Status)
        }

        _, err = io.Copy(os.Stdout, resp.Body)
        if err != nil {
                log.Fatal(err)
        }
}
```

You can download this code here[3].

- Like all Go programs that need to be executed, our program has a package **`main`**.
- Package io[4] provides basic interfaces to I/O primitives.
- On an error we use log[5]. Package **`log`** implements a simple logging package. It defines a type, **`Logger`**, with methods for formatting output. It also has a predefined 'standard' Logger accessible through helper functions **`Print[f|ln]`**, **`Fatal[f|ln]`**, and **`Panic[f|ln]`**, which are easier to use than creating a Logger manually. That logger writes to standard error and prints the date and time of each logged message.
- The **`log.Fatal`** function prints the error message and exits the program.
- For web related http functionality, we import the package net/http[6]. Any functions within that we refer as **`http.function_name`**.

[3] https://gist.github.com/SatishTalim/055387c699bc1af42097
[4] http://golang.org/pkg/io/
[5] http://golang.org/pkg/log/
[6] http://golang.org/pkg/net/http/

- Package os[7] provides a platform-independent interface to operating system functionality.
- In our **`main()`** function, we are setting a variable **`resp`** and doing a **`GET`** request to the Reddit API on our chosen Subreddit.
- The **`func Get(url string) (resp *Response, err error)`** issues a GET to the specified URL. When **`err`** is **`nil`**, **`resp`** always contains a non-nil **`resp.Body`**. Caller should close **`resp.Body`** when done reading from it. The **`resp`** is of type Response[8].
- We use the **`defer`** function to clean up after the HTTP request, and this call will only be executed after the function returns.
- In our Error Handling, check that the HTTP server returns a "200 OK" response. If not, bail, printing the HTTP status message ("500 Internal Server Error", for example).
- The package **`net/http`** defines many constants[9].
- _ is a blank identifier which can be used when we don't care about a particular return value.
- Finally, we copy the **`resp.Body`** (filled with the JSON received from the Reddit API) to the **`os.Stdout`**. The **`resp.Body`** type implements **`io.Reader`** and **`os.Stdout`** implements **`io.Writer`**.

Now you can run the program with the go tool:

```
$ cd $GOPATH/src/github.com/SatishTalim/redditnews
$ go run redditnews.go
```

When you run the program we are outputting (through **`os.Stdout`**) a huge blob of JSON that we received from the Golang Subreddit. Although we can actually see the Articles inside there, this is no good to us. We want to receive the Article's Title, Author's name, a Link to the Article, and we want to assess the value of the article, based on the Reddit link Score the Article has received.

[7] http://golang.org/pkg/os/
[8] http://golang.org/pkg/net/http/#Response
[9] http://golang.org/pkg/net/http/#pkg-constants

4. redditnews.go (Second Iteration)

As seen before, the Reddit API[1] returns JSON data like this:

```
{"data":
  {"children": [
    {"data": {
        "title": "The Go homepage",
        "url": "http://golang.org",
        ...
    }},
    ...
]}}
```

Go's **`json`** package (**`encoding/json`**) decodes JSON-encoded data into native Go data structures. To decode the API response, declare some types that reflect the structure of the JSON data:

```
type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data1 struct {
                Children []struct {
                        Data2 Item `json:"data"`
                } `json:"children"`
        } `json:"data"`
}
```

We are using *tags* on **`struct`** field declarations to customize the encoded JSON key names. The fields use the **`json:`** tag to specify which field they map to.

In Go, top-level declarations beginning with an uppercase letter are "exported" (visible outside the package) and with a lowercase are "unexported" (it can't be directly accessed from any other package). We have kept **`response`** as lowercase. Read William Kennedy's blog post on Exported/Unexported Identifiers In Go[2].

Now let's modify our existing **`main()`** function to **`Decode`** the response into our Data Structure, instead of copying it to **`os.Stdout`**.

[1] http://www.reddit.com/dev/api

[2] http://www.goinggo.net/2014/03/exportedunexported-identifiers-in-go.html

Replace:

```
_, err = io.Copy(os.Stdout, resp.Body)
if err != nil {
        log.Fatal(err)
}
```

With:

```
r := new(response)
err = json.NewDecoder(resp.Body).Decode(r)
```

What we're doing here is initialising a new **`response`** value, and storing a pointer to it in our newly assigned variable **`r`**. Then create a new **`json.Decoder`** object and decode the response body into **`r`**.

We should use **`json.Decoder`** if the data is coming from an **`io.Reader`** stream (for the case of reading from an HTTP request we are obviously reading from a stream), or if we need to decode multiple values from a stream of data.

As the decoder parses the JSON data it looks for corresponding fields of the same names in the **`response`** struct. The "data" field of the top-level JSON object is decoded into the **`response`** struct's **`Data1`** field, and JSON array children are decoded into **`Children`** slice, and so on.

We need to now print the data. To do this, we are going to create a **`for`** loop to iterate over the **`Children`** slice, assigning the slice value to "child" on each iteration. Then we're going to print item's **`Author`**, **`Score`**, **`URL`** and **`Title`** followed by a new line:

```
for _, child := range r.Data1.Children {
        fmt.Println(child.Data2.Author)
        fmt.Println(child.Data2.Score)
        fmt.Println(child.Data2.URL)
        fmt.Println(child.Data2.Title)
}
```

You could use **`log.Println`** instead of **`fmt.Println`**.

Also note that we need to remove the following imports **`io`**, **`os`** and add the following **`encoding/json`**, **`net/http`** to the program that follows.

The complete program so far:

Program redditnews.go

```
package main

import (
        "encoding/json"
        "fmt"
        "log"
        "net/http"
)

type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data1 struct {
                Children []struct {
                        Data2 Item `json:"data"`
                } `json:"children"`
        } `json:"data"`
}

func main() {
        resp, err := http.Get("http://reddit.com/r/golang.json")
        if err != nil {
                log.Fatal(err)
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                log.Fatal(resp.Status)
        }

        r := new(response)
        err = json.NewDecoder(resp.Body).Decode(r)
        if err != nil {
                log.Fatal(err)
        }
```

```
        for _, child := range r.Data1.Children {
                fmt.Println(child.Data2.Author)
                fmt.Println(child.Data2.Score)
                fmt.Println(child.Data2.URL)
                fmt.Println(child.Data2.Title)
        }
}
```

You can download this code here[3].

To work with some printing functions, we import the package **fmt** above.

Now you can run the program with the go tool:

```
$ cd $GOPATH/src/github.com/SatishTalim/redditnews
$ go run redditnews.go
```

Here is a sample output:

```
natefinch
26
http://blog.natefinch.com/2014/05/intro-to-go-interfaces.html
Intro++ to Go Interfaces
Feribg
19
http://www.techtalkshub.com/go-circuit-towards-elastic-computation-failures/
The Go Circuit: Towards Elastic Computation with No Failures
```

[3] https://gist.github.com/SatishTalim/51f8d01cb545ee389dcb

5. redditnews.go (Third Iteration)

So far, all the action happens in the **`main`** function. As the program grows, structure and modularity become important. What if we want to check several subreddits?

Create a function named **`Get`** that takes the name of subreddit, makes the API call, and returns the items from that subreddit.

```
func Get(reddit string) ([]Item, error) {}
```

The function **`Get`** takes an argument called reddit that is a **`string`**. It also returns an **`[]Item`** slice and an **`error`** value (also a **`string`**.)

Here's the code for our **`Get`** function:

```
func Get(reddit string) ([]Item, error) {
        url := fmt.Sprintf("http://reddit.com/r/%s.json", reddit)
        resp, err := http.Get(url)
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, errors.New(resp.Status)
        }

        r := new(Response)
        err = json.NewDecoder(resp.Body).Decode(r)
        if err != nil {
                return nil, err
        }

        items := make([]Item, len(r.Data1.Children))
        for i, child := range r.Data1.Children {
                items[i] = child.Data2
        }
        return items, nil
}
```

- Our new **`Get`** function allows us to call the Reddit API request from anywhere in our program, instead of only from our **`main()`** function.

- When we call the **`Get`** function, we will specify which Subreddit to use.
- We use **`fmt.Sprintf`** to construct the request URL from the provided reddit **`string`**.
- Exiting the function, we return a **`nil`** slice and a non-nil **`error`** value, or vice versa.
- The response's **`Status`** field is just a **`string`**; we use the **`errors.New`** function to convert it to an **`error`** value.
- We use the **`make`** function to allocate an **`Item`** slice big enough to store the response data.
- We iterate over the response's **`Children`** slice, assigning each child's **`Data2`** element to the corresponding element in the items slice.

Next, we need to implement a formatted string that returns the Article's Author, Score, URL and Title.

The **`fmt`** package knows how to format the built-in types, but it can be told how to format user-defined types, too.

When you pass a value to the **`fmt.Print`** functions, it checks to see if it implements the **`fmt.Stringer`** interface:

```
type Stringer interface {
        String() string
}
```

Any type that implements a **`String() string`** method is a **`Stringer`**, and the **`fmt`** package will use that method to format values of that type.

Thus the **`Stringer`** interface will allow us to format User-Defined types, such as the "Item" type we're about to define, to format our Subreddit results.

Here's a **`String`** method for the **`Item`** type that returns the Author, Score, URL, Title and a newline:

```
func (i Item) String() string {
            return fmt.Sprintf(
                "Author: %s\nScore: %d\nURL: %s\nTitle: %s\n\n",
                i.Author,
                i.Score,
                i.URL,
                i.Title)
}
```

To print the **`item`** we just pass it to **`Println`**, which uses the provided **`String`** method to format the **`Item`**.

```
fmt.Println(item)
```

Now that we've implemented this, change your **`main()`** function to reflect this:

```
for _, item := range items {
        fmt.Println(item)
}
```

Discussion

range copies the values from the slice you're iterating over making the original value untouchable. We could use pointers or the index instead. How? I would leave that to you to experiment.

Note that we need to add the following import **errors** to the program that follows.

The complete program so far:

Program redditnews.go

```
package main

import (
        "encoding/json"
        "errors"
        "fmt"
        "log"
        "net/http"
)

type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data1 struct {
                Children []struct {
                        Data2 Item `json:"data"`
                } `json:"children"`
        } `json:"data"`
}

func Get(reddit string) ([]Item, error) {
        url := fmt.Sprintf("http://reddit.com/r/%s.json", reddit)
        resp, err := http.Get(url)
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, errors.New(resp.Status)
        }

        r := new(response)
        err = json.NewDecoder(resp.Body).Decode(r)
        if err != nil {
                return nil, err
        }
```

```
        items := make([]Item, len(r.Data1.Children))
        for i, child := range r.Data1.Children {
                items[i] = child.Data2
        }
        return items, nil
}

func (i Item) String() string {
        return fmt.Sprintf(
                "Author: %s\nScore: %d\nURL: %s\nTitle: %s\n\n",
                i.Author,
                i.Score,
                i.URL,
                i.Title)
}

func main() {
        items, err := Get("golang")
        if err != nil {
                log.Fatal(err)
        }

        for _, item := range items {
                fmt.Println(item)
        }
}
```

You can download this code here[1].

Now when we run our program we should see a nicely formatted list of links.

[1] https://gist.github.com/SatishTalim/7082d3c786b0b20ab035

6. Create a package redditnews

All .go files must declare the package that they belong to as the first line of the file excluding whitespace and comments. Packages are contained in a single directory. All .go files in a single directory must declare the same package name.

The convention for naming your package is to use the name of the directory containing it. This has the benefit of making it clear what the package name is when you import it.

Each folder represents a package. Each package is a unit of code that gets compiled into a static library. When a **`main`** function exists, then these packages are linked together, along with the runtime, to create an executable program.

Change **`redditnews.go`** (remember that we don't need to import **`log`** here) to the following:

Program redditnews.go

```
package redditnews

import (
        "encoding/json"
        "errors"
        "fmt"
        "net/http"
)

type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data1 struct {
                Children []struct {
                        Data2 Item `json:"data"`
                } `json:"children"`
        } `json:"data"`
}

func Get(reddit string) ([]Item, error) {
        url := fmt.Sprintf("http://reddit.com/r/%s.json", reddit)
        resp, err := http.Get(url)
```

```
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, errors.New(resp.Status)
        }

        r := new(response)
        err = json.NewDecoder(resp.Body).Decode(r)
        if err != nil {
                return nil, err
        }

        items := make([]Item, len(r.Data1.Children))
        for i, child := range r.Data1.Children {
                items[i] = child.Data2
        }
        return items, nil
}

func (i Item) String() string {
        return fmt.Sprintf(
                "Author: %s\nScore: %d\nURL: %s\nTitle: %s\n\n",
                i.Author,
                i.Score,
                i.URL,
                i.Title)
}
```

Note that there is no **main** function in the code above. You can download this code here[1].

To confirm whether our package works, create a new folder inside your **redditnews** folder named say **reddit**, and copy the following program **reddit.go** file there.

[1] https://gist.github.com/SatishTalim/248c2f777941e31b2319

Program reddit.go

```
package main

import (
        "fmt"
        "github.com/SatishTalim/redditnews"
        "log"
)

func main() {
        items, err := redditnews.Get("golang")
        if err != nil {
                log.Fatal(err)
        }

        for _, item := range items {
                fmt.Println(item)
        }
}
```

You can download this code here[2].

Remember to import the **redditnews** package, and use the **redditnews.** prefix before the **Get** invocation.

redditnews is the name of the library and **reddit** below as that of the command-line client.

Run **reddit.go** to confirm that our package works.

[2] https://gist.github.com/SatishTalim/0cc2f32417df2b9da323

7. Sending an Email

Finally, let us set up GMail to email us the Reddit News using our **redditnews** package.

First, we will create an **Email()** function in **redditnews.go**, that we can call within our **main()** function to include the formatted Articles list inside the Email HTML body:

```
func Email() string {
        var buffer bytes.Buffer

        items, err := Get("golang")
        if err != nil {
                log.Fatal(err)
        }

        // Need to build strings from items
        for _, item := range items {
                buffer.WriteString(item.String())
        }

        return buffer.String()
}
```

What we're doing here is using the:

- **log** package, and
- inbuilt bytes[1] package to build strings from our items which we can then call from the Body section of our scheduled Email. Let's revisit the **main()** function, and actually build the email we are going to send.

The modified **redditnews.go** (remember to add the following imports **bytes** and **log**) file is:

[1] http://golang.org/pkg/bytes/

Program redditnews.go

```
package redditnews

import (
        "bytes"
        "encoding/json"
        "errors"
        "fmt"
        "log"
        "net/http"
)

type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data1 struct {
                Children []struct {
                        Data2 Item `json:"data"`
                } `json:"children"`
        } `json:"data"`
}

func Get(reddit string) ([]Item, error) {
        url := fmt.Sprintf("http://reddit.com/r/%s.json", reddit)
        resp, err := http.Get(url)
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, errors.New(resp.Status)
        }

        r := new(response)
        err = json.NewDecoder(resp.Body).Decode(r)
        if err != nil {
                return nil, err
        }
```

```
        items := make([]Item, len(r.Data1.Children))
        for i, child := range r.Data1.Children {
                items[i] = child.Data2
        }
        return items, nil
}

func (i Item) String() string {
        return fmt.Sprintf(
                "Author: %s\nScore: %d\nURL: %s\nTitle: %s\n\n",
                i.Author,
                i.Score,
                i.URL,
                i.Title)
}

func Email() string {
        var buffer bytes.Buffer

        items, err := Get("golang")
        if err != nil {
                log.Fatal(err)
        }

        // Need to build strings from items
        for _, item := range items {
                buffer.WriteString(item.String())
        }

        return buffer.String()
}
```

You can download this code here[2].

Create a new folder inside your **redditnews** folder named **redditmail**. Here create a file **redditmail.go** as:

[2] https://gist.github.com/SatishTalim/66d3d64959b55866a41a

Program redditmail.go

```
package main

import (
        "github.com/SatishTalim/redditnews"
        "log"
        "net/smtp"
)

func main() {
        to := "satish@joshsoftware.com"
        subject := "Go articles on Reddit"
        message := redditnews.Email()

        body := "To: " + to + "\r\nSubject: " +
                subject + "\r\n\r\n" + message

        auth := smtp.PlainAuth("", "satish.talim", "password", "smtp.gmail.com")
        err := smtp.SendMail(
                "smtp.gmail.com:587",
                auth,
                "satish.talim@gmail.com",
                []string{to},
                []byte(body))
        if err != nil {
                log.Fatal("SendMail: ", err)
                return
        }
}
```

You can download this code here[3].

- Sign up[4] for a free Gmail account, if you don't have one already.
- We use net/smtp[5] package to connect to GMail.
- Call smtp.PlainAuth[6] with your gmail username, password and domain name, and it returns you back an instance of smtp.Auth[7] that you can use to send e-mails.
- If you have setup the 2-Step Verification process in Gmail, then you need to set an application-specific password[8]. Use this password in the above program.
- You can send mail with smtp.SendMail[9].

[3] https://gist.github.com/SatishTalim/2d4a079a365010f527c0
[4] https://accounts.google.com/SignUp
[5] http://golang.org/pkg/net/smtp/
[6] http://golang.org/pkg/net/smtp/#PlainAuth
[7] http://golang.org/pkg/net/smtp/#Auth
[8] https://support.google.com/accounts/answer/185833
[9] http://golang.org/pkg/net/smtp/#SendMail

- SMTP on port 587 uses TLS. SSL and TLS both provide a way to encrypt a communication channel between two computers (e.g. your computer and a server). TLS is the successor to SSL and the terms SSL and TLS are used interchangeably unless youre referring to a specific version of the protocol.

Run the program and you'll get an email with the details. If interested, you could setup a **`cron`** job to send you an email at 6 am everyday as follows:

```
0 6 * * * cd folder_containing_redditmail_executable && ./redditmail
```

We are telling our Cron Job to run at 0 minutes past the 6th hour (6am) every day here. The Cron Job is changing into the directory in which our executable file is located, then simply running the program.

8. Documenting redditnews

Godoc is the Go documentation tool. It reads documentation directory from Go source files. It's easy to keep documentation and code in sync when they live together in the same place.

Here's our **redditnews** package when viewed from **godoc** on the command line:

```
c:\go_projects\go\src>godoc github.com/SatishTalim/redditnews
PACKAGE DOCUMENTATION

package redditnews
    import "github.com/SatishTalim/redditnews"


FUNCTIONS

func Email() string

func Get(reddit string) ([]Item, error)


TYPES

type Item struct {
    Author string `json:"author"`
    Score  int    `json:"score"`
    URL    string `json:"url"`
    Title  string `json:"title"`
}


func (i Item) String() string



SUBDIRECTORIES

        reddit
        redditmail
```

godoc output

To document add a comment directly above their declarations:

```
// Item describes a RedditNews item.
type Item struct {

// Get fetches the most recent Items posted to the specified subreddit.
func Get(reddit string) ([]Item, error){
```

Most importantly, document the package itself by adding a comment to the `package` clause:

```
// redditnews package implements a basic client for the Reddit API.
package redditnews
```

Don't worry about documenting the **`String`** method, as all Go programmers should be familiar with it and its purpose.

The **`godoc`** output for our revised package:

```
c:\go_projects\go\src>godoc github.com/SatishTalim/redditnews
PACKAGE DOCUMENTATION

package redditnews
    import "github.com/SatishTalim/redditnews"

    redditnews package implements a basic client for the Reddit API.


FUNCTIONS

func Email() string
    Email prepares the body of an email

func Get(reddit string) ([]Item, error)
    Get fetches the most recent Items posted to the specified subreddit.


TYPES

type Item struct {
    Author string `json:"author"`
    Score  int    `json:"score"`
    URL    string `json:"url"`
    Title  string `json:"title"`
}
    Item describes a RedditNews item.


func (i Item) String() string



SUBDIRECTORIES

        reddit
        redditmail
```

godoc output

The modified **`redditnews.go`** file is:

Program redditnews.go

```
// redditnews package implements a basic client for the Reddit API.
package redditnews

import (
        "bytes"
        "encoding/json"
        "errors"
        "fmt"
        "log"
        "net/http"
)

// Item describes a RedditNews item.
type Item struct {
        Author string `json:"author"`
        Score  int    `json:"score"`
        URL    string `json:"url"`
        Title  string `json:"title"`
}

type response struct {
        Data struct {
                Children []struct {
                        Data Item
                }
        }
}

// Get fetches the most recent Items posted to the specified subreddit.
func Get(reddit string) ([]Item, error) {
        url := fmt.Sprintf("http://reddit.com/r/%s.json", reddit)
        resp, err := http.Get(url)
        if err != nil {
                return nil, err
        }

        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
                return nil, errors.New(resp.Status)
        }

        r := new(response)
        err = json.NewDecoder(resp.Body).Decode(r)
```

```
        if err != nil {
                return nil, err
        }

        items := make([]Item, len(r.Data.Children))
        for i, child := range r.Data.Children {
                items[i] = child.Data
        }
        return items, nil
}

func (i Item) String() string {
        return fmt.Sprintf(
                "Author: %s\nScore: %d\nURL: %s\nTitle: %s\n\n",
                i.Author,
                i.Score,
                i.URL,
                i.Title)
}

// Email prepares the body of an email
func Email() string {
        var buffer bytes.Buffer

        items, err := Get("golang")
        if err != nil {
                log.Fatal(err)
        }

        // Need to build strings from items
        for _, item := range items {
                buffer.WriteString(item.String())
        }

        return buffer.String()
}
```

You can download this code here[1].

8.1 References

- Commentary[2]

[1] https://gist.github.com/SatishTalim/22d8a0a1a7ca7fbd0119
[2] http://golang.org/doc/effective_go.html#commentary

- Godoc: documenting Go code[3]
- Documenting Go Code With Godoc[4]

[3] http://blog.golang.org/godoc-documenting-go-code
[4] http://www.goinggo.net/2013/06/documenting-go-code-with-godoc.html

9. Pushing to GitHub

Before we do this it is a good idea to use **`go vet`**.

9.1 Use go vet

The **`go vet`** command will double check your code for common errors.

This tool won't keep you from making huge errors in logic, or from creating buggy code. But it does catch some common errors quite nicely. It's a great idea to get in the habit of running **`go vet`** your code base before you commit it to a source repository.

9.2 Make a new repository on GitHub

Make use of the guide - Make a new repository on GitHub[1], to set up your repository (repo). I named it **`redditnews`**.

In your local **`redditnews`** folder, create two files **`README.md`** and **`LICENSE.txt`** as follows:

Program README.md

```
An Internal Project: redditnews
-------------------------------

[![baby-gopher](https://raw.github.com/drnic/babygopher-site/gh-pages/images/babygopher-ba\
dge.png)](http://www.babygopher.org)

These projects specifications were given to a "Baby Gopher".

> "I am keen to be abreast with what's happening in the **Golang** world. To that end,
> we will write a command-line program (**`redditnews.go`**) that fetches and displays
> the latest headlines from the golang page on Reddit.
>
> The program will:
>
> * make an HTTP request to the Reddit API.
> * decode the JSON response into a Go data structure, and
> * display each link's author, score, URL and title.
>
> We will then be building a _bare-bones_ News Reader package (**`redditnews`**) that
> gives us the latest news and headlines from the Golang Sub-Reddit, using Reddit's
```

[1] https://help.github.com/articles/create-a-repo

```
> API.
>

The "Baby Gopher" built this package and documented his progress so that other
"Baby Gophers" could find it easy to understand the _mechanics_ of writing a package
in Go. He/she would now be able to build their own Go packages.
```

You can download this file here[2].

Program LICENSE.txt

```
The MIT License (MIT)

Copyright (c) [2014] [Satish Talim]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

You can download this file here[3].

Then open a Bash shell for your local folder **redditnews** and type:

```
$ git init
$ git add .
$ git commit -m "first commit"
$ git remote add origin https://github.com/SatishTalim/redditnews.git
$ git push -u origin master
```

[2] https://gist.github.com/SatishTalim/ef0ec69c2d0b2ad75a77
[3] https://gist.github.com/SatishTalim/0d7799a5d87312557eb5

10. Golang Dependency Management

10.1 Git Tagging

Go has no central repository for packages. Instead, you include remote packages in your code by importing a URL that points to the package's remote location. Usually, this is a GitHub, BitBucket, or Google Code URL:

```
import "github.com/SatishTalim/redditnews"
             ^         ^            ^
             |         |            |
             |         |            `--------------- Package / Repo name
             |         `---------------------------- Author's handle
             `-------------------------------------- Hosting site
```

Go doesn't support any explicit method for versioning packages.

10.2 Versioning your package

Gustavo Niemeyer[1] has come out with a package gopkg.in[2] for stable APIs for the Go language, which we shall use here.

The advantage of using gopkg.in is that the URL is cleaner, shorter, redirects to the package documentation at godoc.org when opened with a browser, handles git branches and tags for versioning, and most importantly encourages the adoption of stable versioned package APIs.

Note that gopkg.in does not hold the package code. Instead, the go tool is redirected and obtains the code straight from the respective GitHub repository.

Here's the gopkg.in URL for my repo[3].

GitHub repositories that have no version tags or branches are considered unstable, and thus in "v0". What we had uploaded to GitHub was "v0".

Version zero (v0) is reserved for packages that are so immature that offering any kind of API stability guarantees would be unreasonable. This is equivalent to labeling the package as alpha or beta quality, and as such the use of these packages as dependencies of stable packages and applications is discouraged.

Packages should not remain in v0 for too long, as the lack of API stability hinders their adoption, and hurts the stability of packages and applications that depend on them.

Use the **go** tool to automatically check out and install my package:

[1] https://github.com/niemeyer
[2] http://labix.org/gopkg.in
[3] https://gopkg.in/SatishTalim/redditnews.v0

```
$ go get -u gopkg.in/SatishTalim/redditnews.v0
```

The **-u** flag instructs **get** to use the network to update the named packages and their dependencies. By default, **get** uses the network to check out missing packages but does not use it to look for updates to existing packages.

The **go get** command checks out the repository to **$GOPATH/src/github.com/SatishTalim/redditnews** and installs the binary, if any, to **$GOPATH/bin**. The **bin** directory is in my **PATH**.

The **go get** command can fetch code from:

- Bitbucket
- GitHub
- Google Code
- Launchpad

as well as arbitrary Git, Mercurial, Subversion, and Bazaar repositories.

To import my package, add the following line to your code:

```
import "gopkg.in/SatishTalim/redditnews.v0"
```

Note: Although a version selector (v0) is provided as part of the import path, your code should refer to the Go package as **redditnews**.

10.3 When to change the version?

Examples of modifications that DO NEED a major version change are:

- Removing or renaming *any* exposed name (function, method, type, etc.)
- Adding, removing or renaming methods in an interface
- Adding a parameter to a function, method, or interface
- Changing the type of a parameter or result in a function, method, or interface
- Changing the number of results in a function, method, or interface
- Some struct changes: A struct consisting only of a few exported fields must not have fields added (exported or not) or repositioned without a major version change, as that will break code using such structs in literals with positional fields. Also, removing or renaming exported fields from an existing struct means a major version change.

On the other hand, the following modifications are FINE WITHOUT a major version change:

- Adding exposed names (function, method, type, etc)
- Renaming a parameter or result of a function, method, or interface
- Some struct changes: A struct containing non-exported fields may always receive new exported fields safely, as the language disallows code outside the package from using literals of these structs without naming the fields explicitly.

10.4 How to change the version?

Once our package is stable, we need to change the version number to say "v1". Increasing the version number is done by creating a git tag or branch with the proper name and pushing it, as follows:

```
$ git add -A
$ git commit -m "commit"
$ git tag -a v1 -m 'my version 1.0'
```

The -m specifies a tagging message, which is stored with the tag. If you don't specify a message for an **annotated** tag, Git launches your editor so you can type it in. Annotated tags are stored as full objects in the Git database. They're checksummed; contain the tagger name, e-mail, and date; have a tagging message; and can be signed and verified with GNU Privacy Guard (GPG). It's generally recommended that you create annotated tags so you can have all this information.

By default, the **`git push`** command doesn't transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. Thus:

```
$ git push origin v1
```

10.5 What next?

We have seen the mechanics of building a bare-bones **`redditnews`** package. You can build further upon this:

- Expand the reddit package to support more of the Reddit API.
- Learn about Go's concurrency primitives and perform multiple requests in parallel.

That's it, have fun!