Tools for working with Go Code

Fatih Arslan - @ftharsln

Go was designed to make tools easy to write

Rob Pike

Development cycle

- Formatting
- Navigation & Insight
- Refactor
- Code generation
- Linters & Checkers
- Test and Benchmark
- Distribution & Dependency Management

Formatting

Erik Hollensbe
@erikhollensbe

I wish gofmt could fmt my english too

Ryan Uber
@ryanuber_

Sitting here writing terrible bash, repeatedly saving in vim, thinking "why is it not indenting?". Gofmt has definitely spoiled me.

Community Opinion
@cemerick

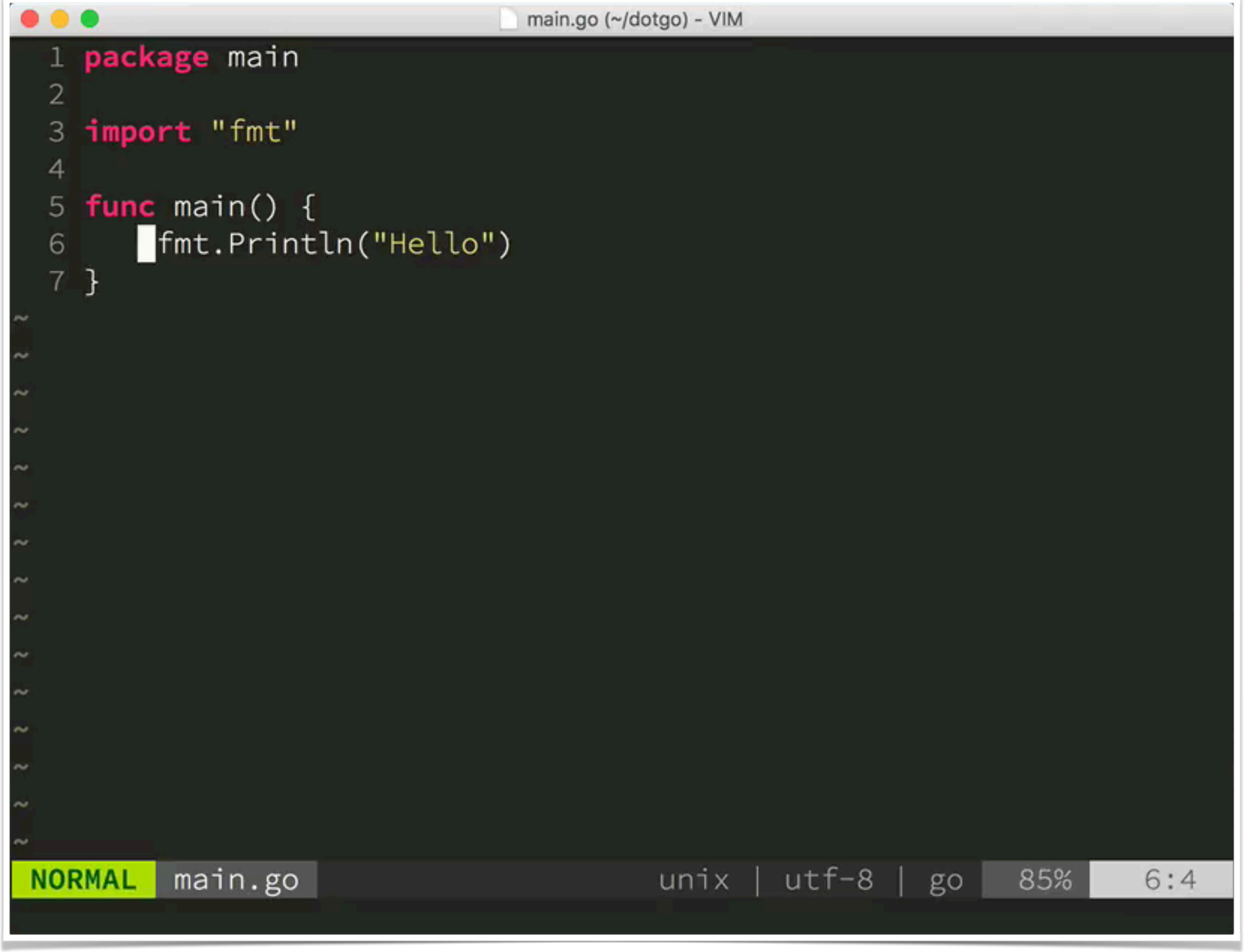
Pike's law: As a #golang discussion grows longer, the probability that gofmt will be used in response to language complaints approaches 1.

goimports

updates your Go import lines

add missing ones
remove unreferenced ones.

play.golang.org has support for goimports too!



Navigation & Insight

godef

prints the location of the symbol referred to

via expression:

```
$ godef -f main.go fmt.Println
/usr/local/Cellar/go/1.5/libexec/src/fmt/print.go:263:6
```

via file offset:

```
$ godef -f main.go -o=50
/usr/local/Cellar/go/1.5/libexec/src/fmt/print.go:263:6
```

How is this useful?

godef

main.go

fmt/print.go

os/file.go

main.go (~/src/dotgo) - VIM

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello DotGo!")
7 }
```

NORMAL main.go unix | utf-8 | go 100% 7:1

dotgo

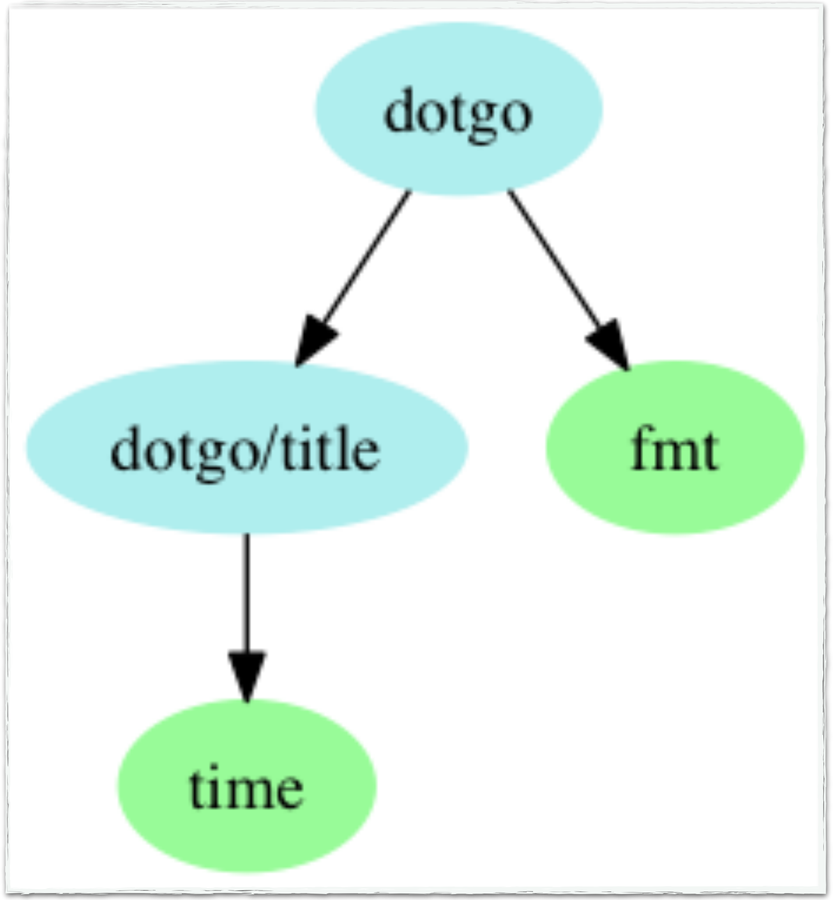
```
package main

import (
   "dotgo/title"
   "fmt"
)

func main() {
   fmt.Println("Hello")
   fmt.Println(title.Name)
}
```

dotgo/title

```
package title

import "time"

var Name = "DotGo! " + time.Hour.String()
```

godepgraph

dependency graph visualization tool



oracle

*a tool for answering questions about Go source code (will be renamed to **guru**)*

Answers questions like:

- What interfaces does this type satisfy?
- Where is the definition of this identifier?
- What are the possible callers of this function?
- What are the exported members of this imported package?

More detail: https://golang.org/s/oracle-user-manual

oracle (cont.)

callers
callees
implements
freevars
…

via file offset:

```
oracle -pos=main.go:#124 mode <pkgs> | <*.go>
```

example usage:

```
$ oracle -format plain -pos='main.go':#151 callers ‘dotgo/'

main.go:11:6: dotgo/.endLines is called from these 1 sites:
main.go:7:19:   static function call from dotgo/.main
```

oracle - callees mode

Answers the questions: *What are the possible targets of this function call?*

```
package example

func DotGo() string {
   return "DotGo"
}

func Colombia() string {
   return "Colombia"
}
```

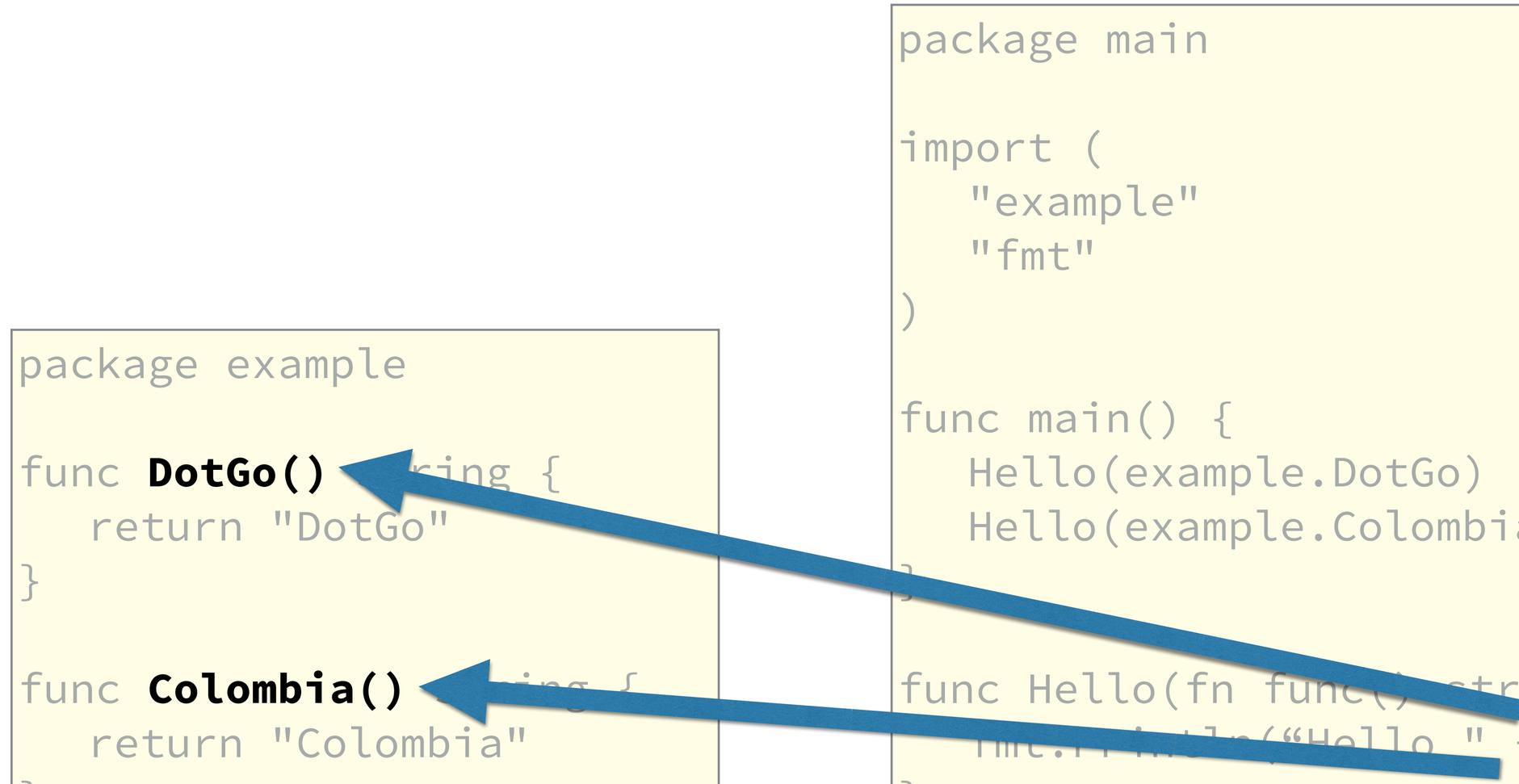
```
package main

import (
   "example"
   "fmt"
)

func main() {
   Hello(example.DotGo)
   Hello(example.Colombia)
}

func Hello(fn func() string) {
   fmt.Println(“Hello " + fn())
}
```

oracle - callees mode

Answers the questions: *What are the possible targets of this function call?*

```
package example

func DotGo() string {
   return "DotGo"
}

func Colombia() string {
   return "Colombia"
}
```

```
package main

import (
   "example"
   "fmt"
)

func main() {
   Hello(example.DotGo)
   Hello(example.Colombia)
}

func Hello(fn func() string) {
   fmt.Println("Hello " + fn())
}
```



```go
package main

import "fmt"

func main() {
   msg := "Greetings\nfrom\nTurkey\n"

   var count int
   for i := 0; i < len(msg); i++ {
      if msg[i] == '\n' {
         count++
      }
   }

   fmt.Println(count)
}
```

oracle - freevars mode

Questions:

What are the free variables of the selected block of code?

```
package main

import "fmt"

func main() {
   msg := "Greetings\nfrom\nTurkey\n"

   var count int
   for i := 0; i < len(msg); i++ {
      if msg[i] == '\n' {
         count++
      }
   }

   fmt.Println(count)
}
```

oracle - freevars mode

Questions:

What are the free variables of the selected block of code?

Can we reuse this functionality somewhere else?

```
package main

import "fmt"

func main() {
   msg := "Greetings\nfrom\nTurkey\n"

   var count int
   for i := 0; i < len(msg); i++ {
      if msg[i] == '\n' {
         count++
      }
   }

   fmt.Println(count)
}
```

oracle - freevars mode

Questions:

What are the free variables of the selected block of code?

Answer:

```
var msg string
```

```go
package main

import "fmt"

func main() {
   msg := "Greetings\nfrom\nTurkey\n"
   fmt.Println(newLines(msg))
}

func newLines(msg string) int {
   var count int
   for i := 0; i < len(msg); i++ {
      if msg[i] == '\n' {
         count++
      }
   }
   return count
}
```

oracle - freevars mode

Refactored the selection into a new function: **newLines()**

The function argument is the free variable named: **msg**

Navigation & Insight (more tools)

- godoc: extracts and generates documentation for Go programs. (it's different than `go doc`). It has command-line support too, such as `go doc`, though the support for command-line usage might be dropped.
- callgraph: display the the call graph of a Go program.
- digraph: queries over directed graphs in text form.
- godex: prints (dumps) exported information of packages or selected package objects.
- ssadump: a tool for displaying and interpreting the SSA form of Go programs.
- gocode: autocompletion daemon for installed Go files.
- gotags: ctags compatible tag generator for Go

Refactoring

gorename

renames identifiers across all packages under $GOPATH

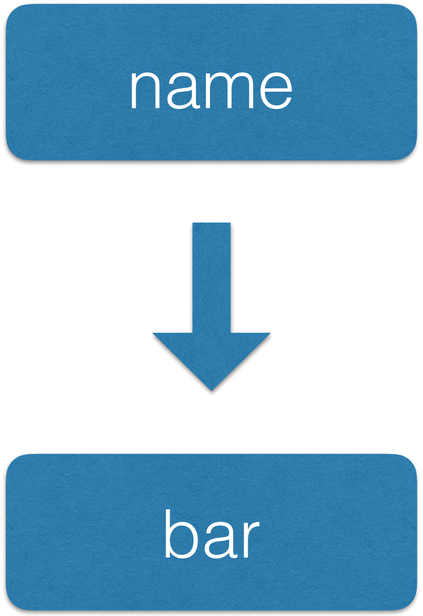
via a query notation:

```
$ gorename -from main.go::name -to identifier
```

via offset (better):

```
$ gorename -offset main.go:#123 -to identifier
```

gorename



main.go (~/Code/koding/go/src/dotgo) - VIM

```
 1 package main
 2
 3 import "fmt"
 4
 5 type Server struct {
 6     name string
 7 }
 8
 9 func main() {
10     s := Server{name: "Kenya"}
11     fmt.Println(s.name)
12 }
~
~
~
~
~
~
~
~
~
~
~
```

NORMAL main.go unix | utf-8 | go 91% 11:23

gomvpkg

moves a package, updating import declarations

Usage:

```
$ gomvpkg -from github.com/fatih/foo -to github.com/fatih/bar
```

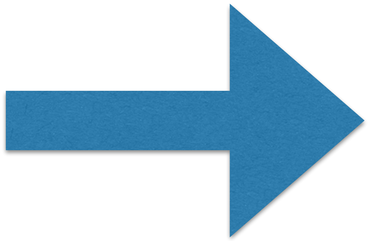
Supports custom move command:

```
$ gomvpkg -from github.com/fatih/foo -to github.com/fatih/bar
          -vcs_mv_cmd "git mv {{.Src}} {{.Dst}}"
```

$ gomvpkg -from bar -to foo

```
src ❯ tree
.
├── bar
│   └── example.go
└── dotgo
    └── main.go
```

```
package main

import (
   "bar"
   "fmt"
)

func main() {
   fmt.Println(bar.T)
}
```



```
src ❯ tree
.
├── dotgo
│   └── main.go
└── foo
    └── example.go
```

```
package main

import (
   "fmt"
   "foo"
)

func main() {
   fmt.Println(foo.T)
}
```

Refactoring (other tools)

- fiximports: rewrite import paths to use canonical package names
- eg: an example-based refactoring tool
- bundle: concatenates the source files of a package to include it into other packages
- Oracle's **freevars** mode (partially)

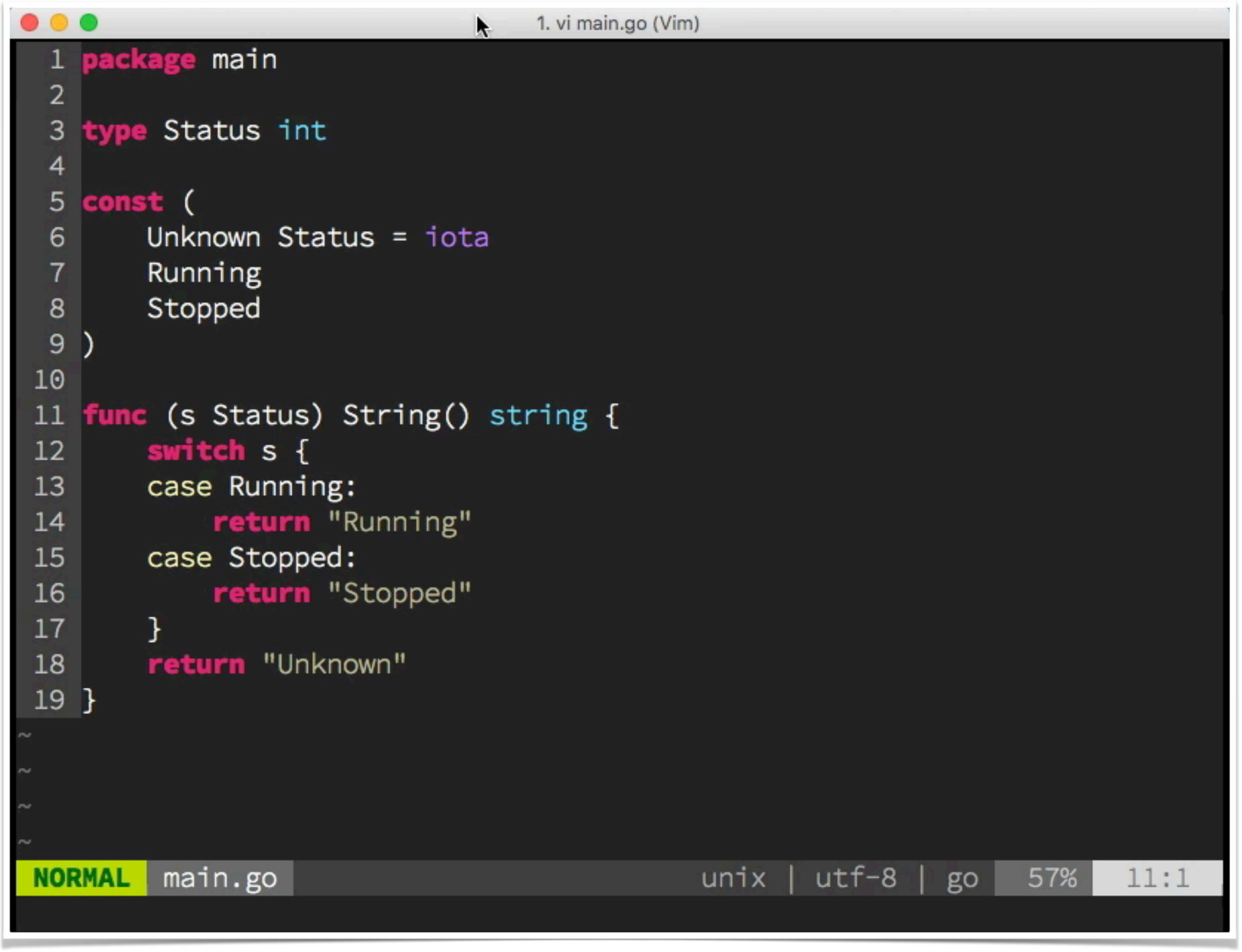
Code Generation

stringer

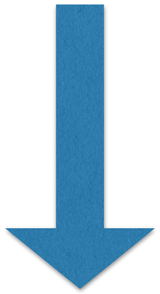
generates String() methods

satisfies fmt.Stringer

```
usage:

stringer -type T
```



```
$ jsonenums -type Status
```



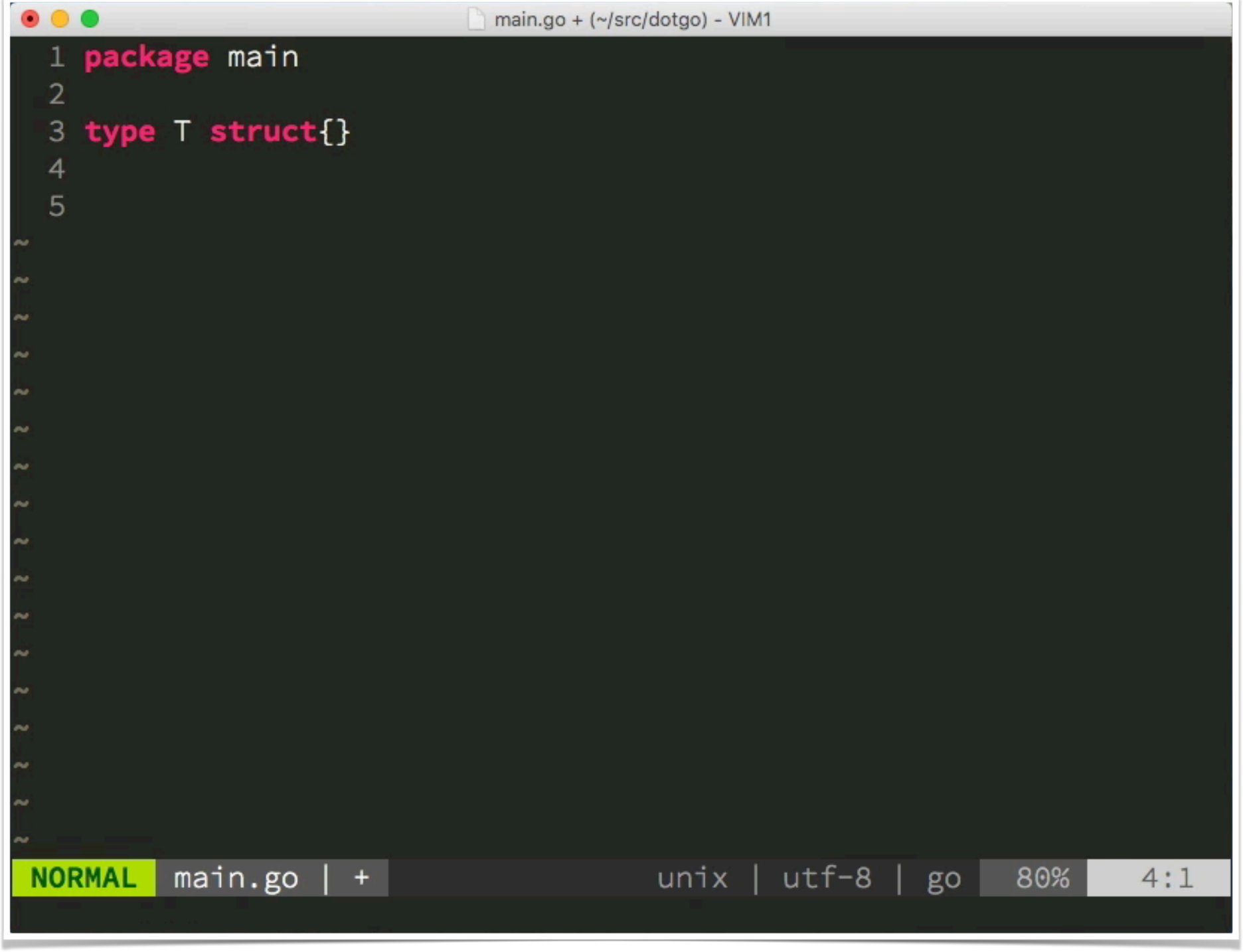
status_jsonenums.go

```
func (r Status) MarshalJSON() ([]byte, error)     {}
func (r *Status) UnmarshalJSON(data []byte) error {}
```

impl

generates
method stubs
for implementing
an interface

```
usage:

impl <recv> <iface>
```



Code generation (more tools)

- gojson: generate golang struct definitions from example JSON
- gen, gotemplate, etc..: generate data structures based on templates (i.e set, slice, list, etc..)
- gene: bootstrapping a Go application from scratch
- sqlgen: generating sql scanners, sql statements and other helper functions
- becky: Asset embedding in to your Go source code
- safekeeper: replace substitute tokens with ENV variables value.
- joiner: generic strings.Join implementation
- optioner: a tool to generate functional options

Linters and checkers

```
//+buildlinux

package main

import (
   "fmt"
   "io"
)

type T struct {
   Foo string `json:"foo`
}

func (t *T) WriteTo(r io.Writer) error {
   return nil
}

func (t *T) Bar() []byte {
   return []byte{}
}
```

How many errors can you detect?

```
func main() {
   fmt.Println("%s", "Hello")

   fmt.Sprintf("%s", "DotGo")



   t := T{}
   if t.Bar != nil {
   }



   return
   fmt.Println("exit")
}
```

How many errors can you detect?

```
//+buildlinux
```
Badly formed build tags

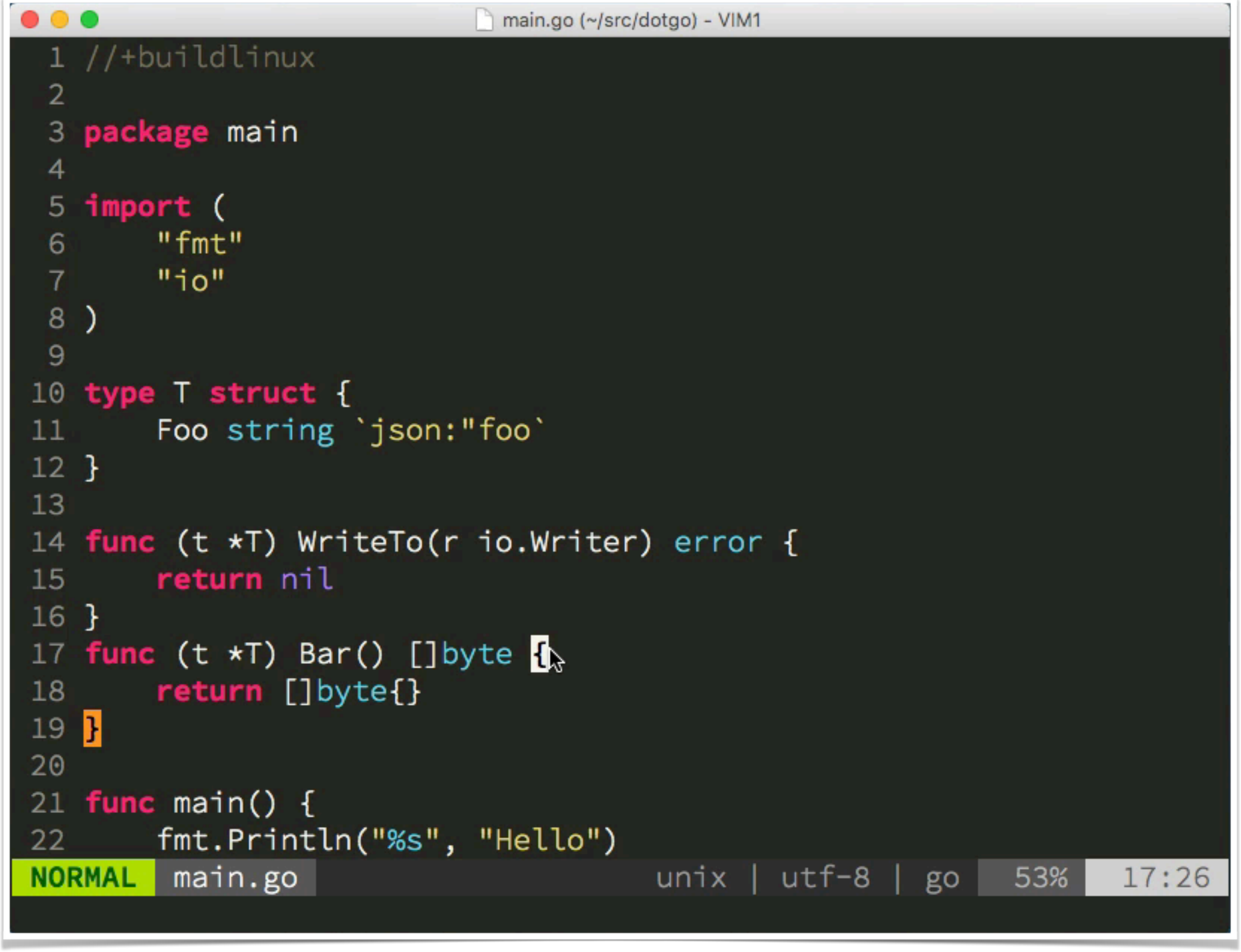
```

package main

import (
   "fmt"
   "io"
)

type T struct {
```
Field tag's quote is not closed
```
   Foo string `json:"foo`
}
```
WriteTo should return the # of bytes written
```
func (t *T) WriteTo(r io.Writer) error {
   return nil
}

func (t *T) Bar() []byte {
   return []byte{}
}
```

Should be fmt.Printf

```
   fmt.Println("%s", "Hello")
```
Return value is not used
```
   fmt.Sprintf("%s", "DotGo")


   t := T{}
```
Should be t.Bar()
```
   if t.Bar != nil {
   }


   return
```
unreachable code
```
   fmt.Println("exit")
}
```

go vet

editor integration



gometalinter

concurrently runs go lint tools and normalize their output

We have to many checkers and linters:

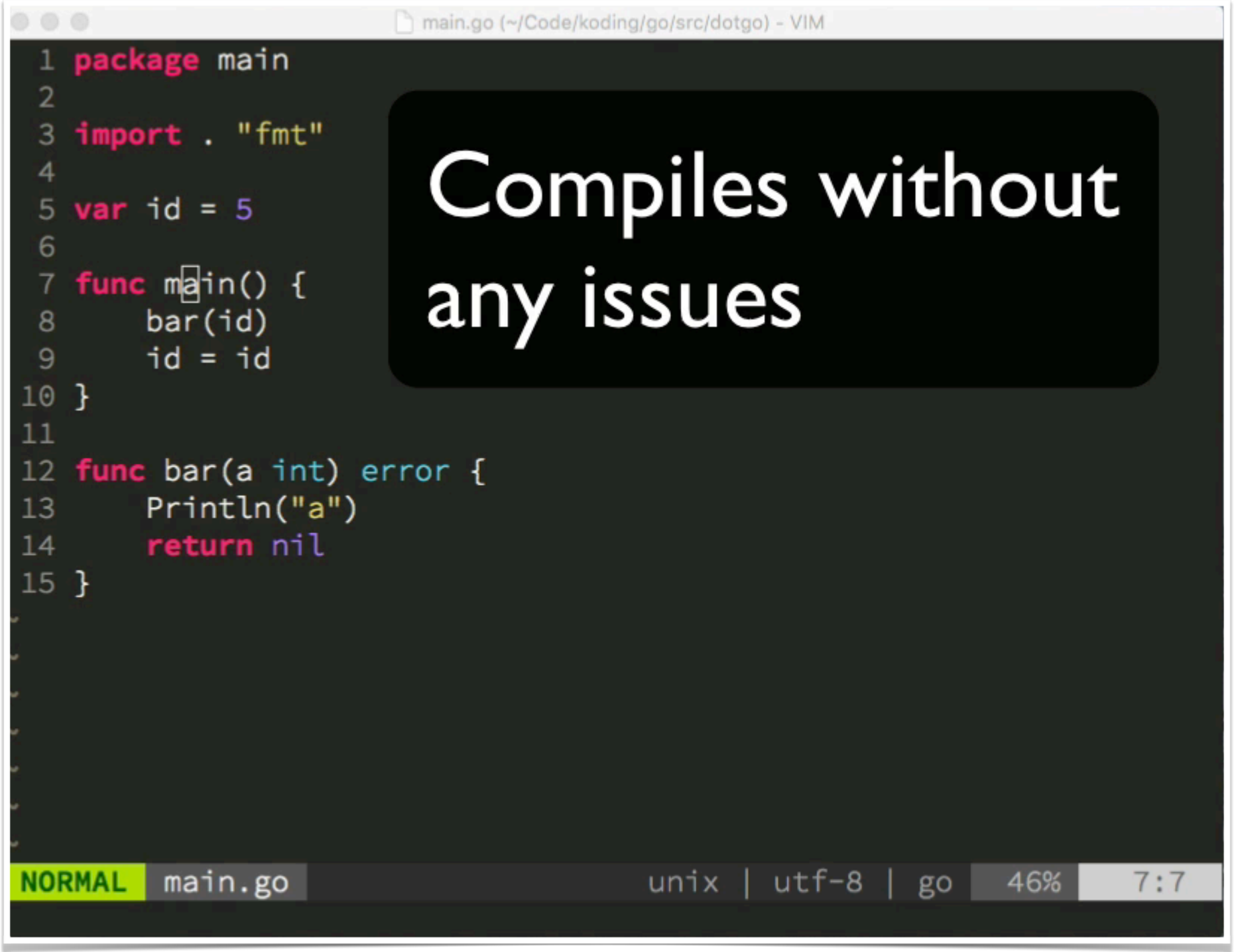
- gotype
- golint
- errcheck
- structcheck
- deadcode
- …

How do we run all of them ?

gometalinter

Let us call these tools:

go vet
errcheck
golint



List of linters and checkers

- gotype: syntactic and semantic analysis of Go files (similar to the front-end of Go compiler). No need to `go build` or `go install` files to show analysis errors.
- golint: linter for Go source code
- errcheck: check for unchecked errors (i.e: func foo() error is used as foo() instead of err := foo())
- structcheck: find unused struct fields
- unexport: unexports unused identifiers (was part of Golang Challenge #5)
- gometalinter: executes checkers and linters, combines the result

Testing and benchmark

- go test: test packages. Enables us to do also benchmark, profiling, coverage …
- benchcmp: benchcmp compares old and new for each benchmark. Add the result as commit message
- cover: a program for analyzing the coverage profiles generated by 'go test -coverprofile=cover.out'. Moved to standard repository with Go release 1.5.
- stress: is intended for catching of episodic failures. It runs a given process in parallel in a loop and collects any failures.
- go-fuzz: parsing of complex inputs (github.com/dvyukov/go-fuzz)

benchcmp

compares old and new for each benchmark.

```
$ go test -run=NONE -bench=. ./... > old.txt

# make changes
$ go test -run=NONE -bench=. ./... > new.txt
```

```
$ benchcmp old.txt new.txt
benchmark              old ns/op     new ns/op     delta
BenchmarkConcat        523           68.6          -86.88%

benchmark              old allocs    new allocs    delta
BenchmarkConcat        3             1             -66.67%
```

Distribution and Dependency Management

- /vendor folder (not an external tool but worth mentioning)
  - Supported by tools like Godeps, Govendor, Glide, gvt, etc..
- godep
- gb (has its own rules)
- More: https://github.com/golang/go/wiki/PackageManagementTools

Editor integrations

We have many great editor integrations:

- vim-go
- go-mode.el (emacs)
- go-plus (atom)
- GoSublime (a better, Google sponsored plugin is being built.)

- LiteIDE
- IntelliJ Idea Plugin
- Eclipse Plugin
- More: https://github.com/golang/go/wiki/IDEsAndTextEditorPlugins


main.go (~/dotgo) - VIM
vim-go support:
NORMAL main.go unix | utf-8 | go 85% 6:30
:Go
GoBuild
GoCallees
GoCallers
GoCallstack
GoChannelPeers
GoCoverage
GoDef
GoDeps
GoDescribe
GoDoc
GoDocBrowser
GoDrop
GoErrCheck
GoFiles
GoFmt
GoFreevars
GoGenerate
GoImpl
GoImplements
GoImport
GoImportAs
GoImports
GoInfo
GoInstall
GoInstallBinaries
GoLint
GoMetaLinter
GoOracleScope
GoPath
GoPlay
GoReferrers
GoRename
GoRun
GoTest
GoTestCompile
GoTestFunc
GoUpdateBinaries
GoVet
:

How to build our own tool?

Lexer and Parser family:

- go/{token, scanner, ast, parser}

Type checker and abstractions

- go/types
- go/ssa (Static Single Assignment)

Builder and formatters:

- go/build
- go/format
- go/printer

Thanks!