Learn the Go Programming Language

For experienced developers or
those of an adventurous nature

gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

Lesson 10

Testing

v0.2 draft

Prelude

- You run tests with `go test`
- http://golang.org/pkg/testing/
- http://golang.org/doc/code.html#Testing
  - How to write go code

GO TEST

```
$ go test
PASS

$ go test -v
=== RUN TestXXX
--- PASS: TestXXX (0.00 seconds)
=== RUN TestYYY
--- FAIL: TestYYY (0.00 seconds)
        filename.go:6: comment
=== RUN TestZZZ
--- SKIP: TestZZZ (0.00 seconds)
        filename.go:45: comment
FAIL
exit status 1
FAIL    /Users/gotutorial/example      0.007s
```

TESTING

TESTING

- http://golang.org/pkg/testing/

```
package main

import "testing"

func TestXxx(t *testing.T) {
    …
    if x == bad {
        t.Error(“expected x is good, but got”, x)
    }
    if y == 0 {
        t.Error(“expected non-zero y, got”, y)
    }
}
func TestYyy(t *testing.T) {
    t.Fatal(“This will always fail")
}
```

TESTING

- http://golang.org/pkg/testing/

pay attention to this “t”

```
package main

import "testing"

func TestXxx(t *testing.T) {
    …
    if x == bad {
        t.Error(“expected x is good, but got”, x)
    }
    if y == 0 {
        t.Error(“expected non-zero y, got”, y)
    }
}
func TestYyy(t *testing.T) {
    t.Fatal(“This will always fail")
}
```

THE "T"

```
func (t *T) Error(args ...interface{})
func (t *T) Errorf(format string, args ...interface{})
func (t *T) Fail()
func (t *T) FailNow()
func (t *T) Failed() bool
func (t *T) Fatal(args ...interface{})
func (t *T) Fatalf(format string, args ...interface{})
func (t *T) Log(args ...interface{})
func (t *T) Logf(format string, args ...interface{})
func (t *T) Parallel()
func (t *T) Skip(args ...interface{})
func (t *T) SkipNow()
func (t *T) Skipf(format string, args ...interface{})
func (t *T) Skipped() bool
```

log & fail

trigger failure

log, fail & end testing

log messages

NOTES

- tests are compiled "inside" a package
    - access to unexpected stuff
- go doesn't do mocks well

```
// Instead of:
func (s *Server)FetchThing()
func doSomething(p *Server)

// Try this:
type error Fetcher {
    FetchThing()
}
func doSomething(fetcher Fetcher)
```

Table Driven Tests

- Very common pattern in go unit tests
- http://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go
- https://github.com/golang/go/wiki/TableDrivenTests

Table Driven Tests

```
var test_table = []struct {
    input  int64
    expected string
}{
    {"1", "one"},
    {"2", "two"},
    {"10", "ten"},
    {"101", "one hundred one"},
    {"1030", "one thousand thirty”},
}

func TestWordify(t *testing.T) {
   for i, case := range test_table {
        candidate := wordify(case.input)
        if candidate != case.expected {
            t.Errorf(i, "got", candidate, "expected", expected)
        }
    }
}
```

SETUP / TEARDOWN

```
// per test setup, teardown
func TestXXX(t *testing.T) {
    setup()
    defer teardown()

    // run your tests
}
```

Global Setup / TearDown

- Use TestMain()
- Only in Go 1.4 and later
- http://cs-guy.com/blog/2015/01/test-main/
- https://blog.unrolled.ca/test-main/

Global Setup / TearDown

```
func TestMain(m *testing.M) {
    global_setup()
    exit_val := m.Run()
    global_teardown()
    os.Exit(exit_val)
}

func TestXXX(t *testing.T) {
   // do your test
}
func TestYYY(t *testing.T) {
   // do your test
}
```

os.Exit() kills the process before any defer executes

Conditional Skipping

```
func (t *T) SkipNow()
func (t *T) Skip(args ...interface{})
func (t *T) Skipf(format string, args ...interface{})
func (t *T) Skipped() bool

func TestWhenServerUp(t *testing.T) {
  if server_is_available() {
    t.Skip("can't get to server")
  }
  // ...
}

// ONLY visible when running go test -v
=== RUN TestWhenServerUp
--- SKIP: TestWhenServerUp (0.00 seconds)
    xyz_test.go:176: can't get to server
```

Short Mode

```
go test -short

func TestTimeConsuming(t *testing.T) {
    if testing.Short() {
        t.Skip("skipping test in short mode.")
    }
    ...
}
```

PARALLELIZATION

- by default, go test will run only one test at a time.
- by default go test will run the tests with the current value of GOMAXPROCS
  - use go test -cpu=1,2,4 to run the test multiple times with different values of GOMAXPROCS

PARALLELIZATION

- You have to manually mark which tests are allowed to run in parallel
- Will run up to GOMAXPROCS tests in parallel

```
package main

import "testing"

func TestXXX(t *testing.T) {
    t.Parallel()
    // ...
}
func TestYYY(t *testing.T) {
    t.Parallel()
    // ...
}
```

RACE DETECTION

- go has a very fancy race detector
- http://blog.golang.org/race-detector
- must be explicitly enabled

```
// not just for test!
go test -race mypkg    // to test the package
go run -race mysrc.go  // to run the source file
go build -race mycmd   // to build the command
```

BMIZERANY/ASSERT

```
// https://github.com/bmizerany/assert
// super handy shortcut

// Instead of

if (reflect.DeepEqual(p1, p2) == false) {
    t.Fatal(“FAIL”, file, line, p1, “!=“, p2)
}

// you can do this:
    assert.Equal(t, p1, p2)
```

BENCHMARKING

BENCHMARKING

- This should look familiar:

```
package main

import "testing"

func BenchXXX(b *testing.B) {

}
func BenchYYY(b *testing.B) {

}
```

BENCHMARKING

```
package main

import "testing"

func BenchXxx(b *testing.B) {
    for i := 0; i < b.N; i++ {
        fmt.Sprintf("hello")
    }
}
```

Bench Example

```
func runHasher(b *testing.B, hasher hash.Hash) {
  inputs := [][]byte{
    []byte(“a"),
    []byte(“abcdefghijklmnopqrztuvwxyz012345”),
  }
  for i := 0; i < b.N; i++ {
    for _, input := range inputs {
      hasher.Write(input)
      hasher.Sum(nil)
    }
  }
}

func BenchmarkSHA1(b *testing.B) {
  runHasher(b, sha1.New())
}
func BenchmarkSHA256(b *testing.B) {
  runHasher(b, sha256.New())
}
func BenchmarkSHA512(b *testing.B) {
  runHasher(b, sha512.New())
}
```

BENCH EXAMPLE

```
func runHasher(b *testing.B, hasher hash.Hash) {
  inputs := [][]byte{
    []byte(“a"),
    []byte(“abcdefghijklmnopqrztuvwxyz012345”),
  }
  for i := 0; i < b.N; i++ {
    for _, input := range inputs {
      hasher.Write(input)
      hasher.Sum(nil)
    }
  }
}
```

how many N iterations

output:

```
$ go test -bench .
PASS
BenchmarkSHA1    100000     29730 ns/op
BenchmarkSHA256   20000     75560 ns/op
BenchmarkSHA512   50000     46468 ns/op
ok  github.com/amattn/gobench/shabench 13.449s
```

average duration of a single iteration

Coverage

Coverage

- Test coverage tool built into go as of version 1.2
- Install:

```
go get code.google.com/p/go.tools/cmd/cover
```

- In the simple case, will just spit out a %

```
$ go test -cover
PASS
coverage: 42.9% of statements
ok      size    0.026s
```

COVERAGE BY FUNCTION

- Getting fancy, we check coverage by function:

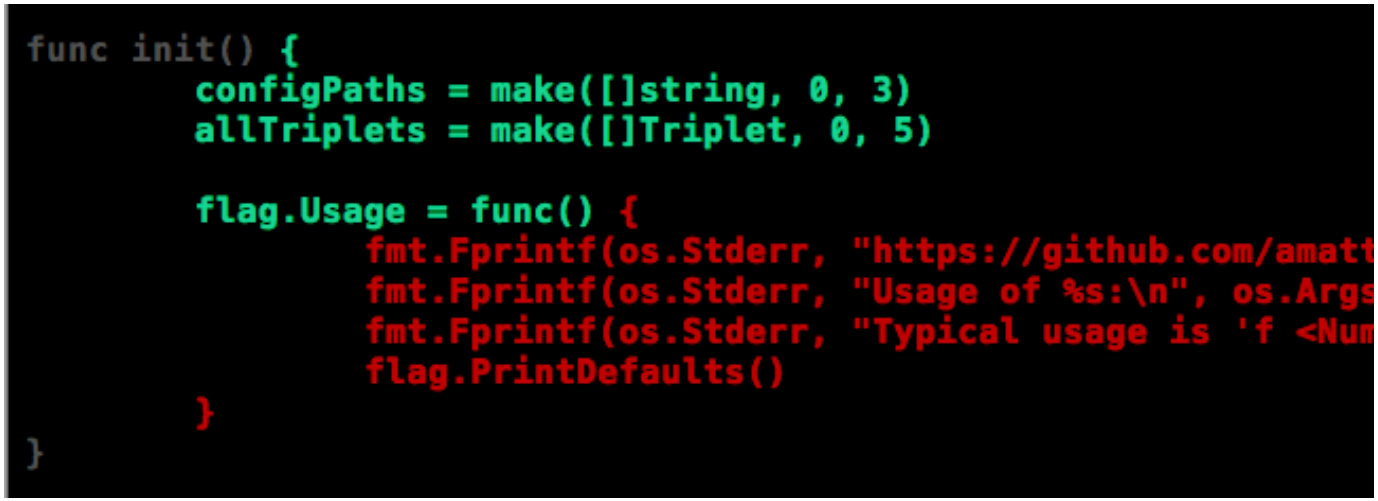
```
$ go test -coverprofile=coverage.out
$ go tool cover -func=coverage.out

github.com/amattn/f/config.go:21:init42.9%
github.com/amattn/f/config.go:43:prependConfigPath0.0%
github.com/amattn/f/config.go:53:appendConfigPath66.7%
github.com/amattn/f/config.go:60:joinFrcToDir100.0%
github.com/amattn/f/config.go:205:cleanLine83.3%
github.com/amattn/f/main.go:15:init100.0%
github.com/amattn/f/main.go:23:main0.0%
github.com/amattn/f/triplet.go:20:IsValid0.0%
github.com/amattn/f/triplet.go:24:IsEqual66.7%
<SNIP>
github.com/amattn/f/triplet.go:89:PrintMenu0.0%
total:(statements)21.1%
```

Coverage Browser Output

- Getting extra fancy, generate color coded html output:

```
$ go test -coverprofile=coverage.out
$ go tool cover -html=coverage.out
```



Coverage Heatmap Output

- Getting extra, extra fancy, generate heat maps:

```
$ go test -covermode=count -coverprofile=count.out
$ go tool cover -html=count.out
```

Coverage: More reading

- http://blog.golang.org/cover
- http://dave.cheney.net/2013/11/14/more-simple-test-coverage-in-go-1-2
  - A nice writeup with some helper scripts

WebApp Testing

NET/HTTP CLIENT

- Shortcuts for GET and POST:

```
resp, err := http.Get("http://example.com/")
resp, err := http.Post("http://example.com/upload",
"image/jpeg", &buf)
```

- PUT, DELETE, et al. use client.Do()

NET/HTTP CLIENT

```
func testHttpClient(t *testing.T) {
    client := new(http.Client)

    var reqData io.Reader = bytes.NewReader([]byte(“request body”))
    method := “GET”
    url := “http://localhost:8080/test_url”

    req, err := http.NewRequest(method, url, reqData)
    if err != nil { t.Fatal(“could not create request”, err) }

    resp, err := client.Do(req)
    if err != nil { t.Fatal("client.Do failure”, err)}

    defer resp.Body.Close()
    bodyBytes, err := ioutil.ReadAll(resp.Body)
    if err != nil { t.Fatal("runTestPath ioutil.ReadAll failure") }

    if resp.StatusCode != expected_status { t.Error(“…”)}
}
```

NET/HTTP CLIENT

```
func testHttpClient(t *testing.T) {
    client := new(http.Client)

    var reqData io.Reader = bytes.NewReader([]byte("request body"))
    method := "GET"
    url := "http://localhost:8080/test_url"

    req, err := http.NewRequest(method, url, reqData)
    if err != nil { t.Fatal("could not create request", err) }

    resp, err := client.Do(req)
    if err != nil { t.Fatal("client.Do failure", err)}

    defer resp.Body.Close()
    bodyBytes, err := ioutil.ReadAll(resp.Body)
    if err != nil { t.Fatal("runTestPath ioutil.ReadAll failure") }

    if resp.StatusCode != expected_status { t.Error("…")}
}
```

Make a request

client.Do()

read the body & inspect the response

NET/HTTP/HTTPTEST: RESPONSERECORDER

```
func main() {
    handler := func(w http.ResponseWriter, r *http.Request) {
        http.Error(w, "something failed”, 500)
    }

    req, err := http.NewRequest("GET", "http://example.com/",
nil)
    if err != nil {
        log.Fatal(err)
    }

    w := httptest.NewRecorder()
    handler(w, req)
    fmt.Printf("%d - %s", w.Code, w.Body.String())
}
```

NET/HTTP/HTTPTEST: RESPONSERECORDER

define a handler

```
func main() {
    handler := func(w http.ResponseWriter, r *http.Request) {
        http.Error(w, "something failed”, 500)
    }
```

define a request

```
    req, err := http.NewRequest("GET", "http://example.com/",
nil)
    if err != nil {
        log.Fatal(err)
    }
```

define a recorder

```
    w := httptest.NewRecorder()
    handler(w, req)
    fmt.Printf("%d - %s", w.Code, w.Body.String())
}
```

“handle” the request and

inspect the recorded response

NET/HTTP/HTTPTEST:TESTSERVER

```
func TestServeHTTP(t *testing.T) {

    server, err := // something here that implements ServeHTTP

    ts := httptest.NewServer(server)
    defer ts.Close()

    res, err := http.Get(ts.URL)
    if err != nil {
        log.Fatal(err)
    }
    _, err = ioutil.ReadAll(res.Body)
    res.Body.Close()
    if err != nil {
        log.Fatal(err)
    }
}
```

NET/HTTP/HTTPTEST: TESTSERVER

```
func TestServeHTTP(t *testing.T) {

    server, err := // something here that implements ServeHTTP

    ts := httptest.NewServer(server)
    defer ts.Close()

    tps := []TestPath{
        TestPath{"GET", "/post", http.StatusMethodNotAllowed, nil},
        TestPath{"PUT", "/post", http.StatusMethodNotAllowed, nil},
        TestPath{"DELETE", "/post", http.StatusMethodNotAllowed, nil},
        TestPath{"POST", "/", http.StatusNotFound, nil},
        TestPath{"POST", "/asdf", http.StatusNotFound, nil},
        TestPath{"POST", "/post/asdfa", http.StatusNotFound, nil},
        TestPath{"POST", "/post", http.StatusOK, byte[]("posted!")},
    }

    for i, tp := range tps {
    runTestPath(t, i, tp, ts)
    }
}

type TestPath struct {
    Method         string // GET, POST, etc.
    Path           string
    ExpectedStatus int
    RequestBody    []byte
}
```

NET/HTTP/HTTPTEST: TESTSERVER

```
func runTestPath(t *testing.T, i int, tp TestPath, ts *httptest.Server) {
    client := new(http.Client)
    var reqData io.Reader
    if tp.RequestBody != nil && len(tp.RequestBody) > 0 {
        reqData = bytes.NewReader(tp.RequestBody)
    }

    req, err := http.NewRequest(tp.Method, ts.URL + tp.Path, reqData)
    if err != nil { t.Fatal("runTestPath Cannot create request", i, tp) }

    resp, err := client.Do(req)
    if err != nil { t.Fatal("runTestPath client.Do failure", i, tp)}

    defer resp.Body.Close()
    bodyBytes, err := ioutil.ReadAll(resp.Body)
    if err != nil { t.Fatal("runTestPath ioutil.ReadAll failure", i, tp) }

    if resp.StatusCode != tp.ExpectedStatus { t.Error(i, tp.Method, tp.Path,
"expected:", tp.ExpectedStatus, "got:", resp.Status, string(bodyBytes))}

    // do more here or call specific testers
}
```

Testing Misc

IO TESTING

Don’t use these in production. Please.

```
// Induce error
func DataErrReader(r io.Reader) io.Reader
func TimeoutReader(r io.Reader) io.Reader

// Slow
func HalfReader(r io.Reader) io.Reader
func OneByteReader(r io.Reader) io.Reader
func TruncateWriter(w io.Writer, n int64) io.Writer

// Log while doing io to stdout
func NewReadLogger(prefix string, r io.Reader) io.Reader
func NewWriteLogger(prefix string, w io.Writer) io.Writer
```

QUICKCHECK

```
import “testing/quick"

func TestOddMultipleOfThree(t *testing.T) {
  f := func(x int) bool {
    y := OddMultipleOfThree(x)
    return y%2 == 1 && y%3 == 0
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}
```

inject random ints

```
func TestString(t *testing.T) {
  f := func(s string) bool {
    err := ProcessString(s)
    return err == nil
  }
  if err := quick.Check(f, nil); err != nil {
    t.Error(err)
  }
}
```

inject random strings

CI

- not Go by ThoughtWorks (http://go.cd)
  - Java based CI platform with an unfortunate name
- jenkins and dotCI plugin
  - CollectiveHealth uses this
- drone.io
- TravisCI (https://travis-ci.org)
- CircleCI (https://circleci.com)

GO VET & GOLINT

- go vet
  - static analysis, mostly about correctness
  - does not guarantee false positives
- golint
  - mostly about coding style

Testing Misc.

- The best reference or guide for go tests is the test files in the go standard library
- More:
  - https://splice.com/blog/lesser-known-features-go-test/
  - http://talks.golang.org/2014/testing.slide
  - https://blog.golang.org/examples
  - http://labix.org/gocheck
    - richer add-on library
  - http://goconvey.co
    - BDD

Thank You, Credits & License

http://gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

- I owe many many, thanks to the many authors of Go.
- These slides are Copyright 2013-2015 Matthew Nunogawa