Unit Testing in Go

29 October 2015 -- 23 February 2019

Ákos Frohner

Overview

Code:

github.com/szamcsi/golang-elte-2019-public/ (https://github.com/szamcsi/golang-elte-2019-public/)

Examples are in **ex?** subdirectories with skeleton code.

Line Count: use case

Counting the number of lines (\n) in files.

```
$ git clone https://github.com/szamcsi/golang-elte-2019-public.git
$ cd golang-elte-2019-public/testing/ex1
$ go run lc.go -- lc.go
31     lc.go
```

Line Count: code

```
func lineCount(path string) int {
    f, err := os.Open(path)
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()
    s := bufio.NewScanner(f)
    lc := 0
    for s.Scan() {
        lc++
    }
    return lc
}

func main() {
    flag.Parse()
    for _, path := range flag.Args() {
        fmt.Printf("%d\t%s\n", lineCount(path), path)
    }
}
```

Testing in Go

All **Test...** functions are sequentially executed in arbitary order.

golang.org/pkg/testing#T (http://golang.org/pkg/testing#T)

```
func TestFunc(t *testing.T) {
  if err := setup(); err != nil {
    t.Fatal(err)
  }
  if got, want := Func(), 10; got != want {
    t.Errorf("Func() = %d; want = %d", got, want)
  }
}
```

Binaries of the same package (including 'main'):

- lc.go --> lc
- lc.go + lc_test.go --> lc_test

The sources of lc.go and lc_test.go are essentially merged, everything is visible.

Write useful error messages

The test error message should be enough to debug the issue without looking at the test source code. What was called, in what environment, what was the result and the expectation. E.g.

```
t.Errorf("LineCount(%q) = %d; want %d", ...)
```

which results in error message:

```
LineCount("testdata/10") = 11; want 10
```

Assert-pattern messages:

```
lc_test.go:46: failed: 10 != 11
```

are not that useful.

Exercise 1

Exercise 1: testing lineCount()

We have a test case in testdata with 10 lines: `ex1/testdata/10`

- write a test: `ex1/lc_test.go`

  golang.org/pkg/path/filepath/#Join (https://golang.org/pkg/path/filepath/#Join)

- run the test:

```
$ go test lc.go lc_test.go
```

- fix the code: `ex1/lc.go`

Exercise 1 (code)

ex1/lc_test.go

```
package main

import "testing"

const testDir = "../testdata/"

func TestLineCount(t *testing.T) {
    // TODO: use "path/filepath" and testDir to access file "10".
    // TODO: use t.Errorf() to indicate a test failure.
}
```

Solution 1 (code)

```
func TestLineCount(t *testing.T) {
    path := filepath.Join(testDir, "10")
    if got, want := lineCount(path), 10; got != want {
        t.Errorf("lineCount('10') = %d; want = %d", got, want)
    }
}
```

```
func lineCount(path string) int {
    f, err := os.Open(path)
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()
    s := bufio.NewScanner(f)
    lc := 0
    for s.Scan() {
        lc++
    }
    return lc
}
```

Exercise 2

Table Driven Tests

- test case loop
- type declaration: slice of struct
- variable initialization

```
for _, tc := range []struct {
  path  string
  count int
}{
  {"10", 10},
  {"11", 11},
} {
  got := lineCount(tc.path)
  ...
}
```

golang.org/wiki/TableDrivenTests (https://golang.org/wiki/TableDrivenTests)

Exercise 2: table-driven testing of lineCount()

Add another test case with 11 lines and update the test!

`ex2/lc_test.go`

```
func TestLineCount(t *testing.T) {
    // TODO: add test for the file with 11 lines!
    path := "10"
    if got, want := lineCount(filepath.Join(testDir, path)), 10; got != want {
        t.Errorf("lineCount(%q) = %d; want = %d", path, got, want)
    }
}
```

Solution 2

```
func TestLineCount(t *testing.T) {
    for _, tc := range []struct {
        path  string
        count int
    }{
        {"10", 10},
        {"11", 11},
    } {
        if lc := lineCount(filepath.Join(testDir, tc.path)); lc != tc.count {
            t.Errorf("lineCount(%q) = %d; want = %d", tc.path, lc, tc.count)
        }
    }
}
```

Exercise 3

Coverage

blog.golang.org/cover (https://blog.golang.org/cover)

```
$ go test -coverprofile=coverage.out lc.go lc_test.go
ok      command-line-arguments    0.018s    coverage: 66.7% of statements
$ go tool cover -func=coverage.out
lc.go:12:    lineCount    88.9%
lc.go:26:    main         0.0%
```

```
func lineCount(path string) int {
    f, err := os.Open(path)
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()
    s := bufio.NewScanner(f)
    lc := 0
    for s.Scan() {
        lc++
    }
    return lc
}
func main() {
    flag.Parse()
    for _, path := range flag.Args() {
        fmt.Printf("%d\t%s\n", lineCount(path), path)
```

```
    }
}
```

Exercise 3: refactor code for testability (lines.go)

Improve the test coverage by refactoring the code!

- Separate the lineCount() to a non-main package

`ex3/lines/lines.go`

```
package lines

// Count counts the lines in a file, specified by the path.
// It returns an error, if the file was not readable.
func Count(path string) (int, error) {
    // TODO: move main.lineCount() here and return an error.
```

Exercise 3: refactor code for testability

- Don't panic!

ex3/lc.go

```
func lineCount(path string) int {
    f, err := os.Open(path)
    if err != nil {
        log.Fatal(err) // DO NOT PANIC!
    }
    defer f.Close()
    s := bufio.NewScanner(f)
    lc := 0
    for s.Scan() {
        lc++
    }
    return lc
}
```

Exercise 3: test error cases (lines_test.go)

- test the missing file

ex3/lines/lines_test.go

```
func TestCount(t *testing.T) {
    for _, tc := range []struct {
        path  string
        count int
    }{
        {"10", 10},
        {"11", 11},
    } {
        // TODO: check with a not-existing file, if an error is returned from Count().
        if lc, _ := Count(filepath.Join(testDir, tc.path)); lc != tc.count {
            t.Errorf("Count(%q) = %d; want = %d", tc.path, lc, tc.count)
        }
    }
}
```

Solution 3 (lines_test.go)

I/O error is covered with a test case now:

```
func TestCount(t *testing.T) {
    for _, tc := range []struct {
        path    string
        count   int
        wantErr bool
    }{
        {"10", 10, false},
        {"11", 11, false},
        {"12", 0, true},
    } {
        // We do not match the exact error message, because that would make the test fragile.
        lc, err := Count(filepath.Join(testDir, tc.path))
        if (err != nil) != tc.wantErr {
            t.Errorf("Count(%q) err != nil is %v, want %v", tc.path, err != nil, tc.wantErr)
            continue
        }
        if lc != tc.count {
            // Returned value can be valid only if err value was as expected.
            t.Errorf("Count(%q) = %d; want = %d", tc.path, lc, tc.count)
        }
    }
}
```

Subtests

`t.Run` runs a function as a nested named test case.

```
func TestCountSubtests(t *testing.T) {
    for _, tc := range []struct {
        // ...
    } {
        t.Run(tc.path, func(t *testing.T) {
            lc, err := Count(filepath.Join(testDir, tc.path))
            if (err != nil) != tc.wantErr {
                // interrupt only the subtest, not TestCountSubtests
                t.Fatalf("Count(%q) err != nil is %v, want %v", tc.path, err != nil, tc.wantErr)
            }
            if lc != tc.count {
                // Returned value can be valid only if err value was as expected.
                t.Errorf("Count(%q) = %d; want = %d", tc.path, lc, tc.count)
            }
        })
    }
}
```

Exercise 4

"Complex" functionality

- Adding new functionality: minimum and maximum line length.

```
func minMaxCount(lines []string) *MinMaxCount
```

- Also separating I/O to another function:

```
func load(path string) ([]string, error)
```

ex4/lines.go

```
func Count(path string) (*MinMaxCount, error) {
    lines, err := load(path)
    if err != nil {
        return nil, err
    }
    return minMaxCount(lines), nil
}
```

TDD: try writing the tests first!

Exercise 4: comparing structures (ex4/lines_test.go)

Use **cmp.Compare** from

godoc.org/github.com/google/go-cmp/cmp (https://godoc.org/github.com/google/go-cmp/cmp)

$ go get github.com/google/go-cmp/cmp

```
func TestLoad(t *testing.T) {
    for _, tc := range []struct {
        path    string
        lines   []string
        wantErr bool
    }{
        {"10", []string{"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"}, false},
        {"11", []string{"1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11"}, false},
        {"12", nil, true},
    } {
        lines, err := load(filepath.Join(testDir, tc.path))
        if tc.wantErr != (err != nil) {
            t.Errorf("load(%q) err != nil is %v; want %v", tc.path, err != nil, tc.wantErr)
            continue
        }
        if true { // TODO: compare lines and tc.lines
            t.Errorf("load(%q) returned diff (-want +got):\n%s", tc.path,
                "") // TODO: print details of how the result differs from expected
        }
```

```
    }
}
```

Solution 4

```
        if !cmp.Equal(lines, tc.lines) {
            t.Errorf("load(%q) returned diff (-want +got):\n%s", tc.path,
                cmp.Diff(tc.lines, lines))
        }
```

Exercise 5

Exercise 5: faking a package level function? (ex5/stamp.go)

```
// BuildStamp returns a unique build stamp string.
func BuildStamp() string {
    hostname, err := os.Hostname() // TODO: refactor for testing
    if err != nil {
        hostname = "<unknown>"
    }
    now := time.Now() // TODO: refactor for testing
    return fmt.Sprintf("%s@%s", hostname, now.Format(timeFormat))
}
```

```
func TestBuildStamp(t *testing.T) {
    for _, tc := range []struct {
        now      string
        hostname string
        err      error
        want     string
    }{
        {"2015-10-25T10:11:12", "foo", nil, "foo@2015-10-25T10:11:12"},
        {"2015-10-25T10:11:12", "", errors.New("no name"), "<unknown>@2015-10-25T10:11:12"},
    } {
        // TODO: inject time.Now() and os.Hostname()!
        if got := BuildStamp(); tc.want != got {
            t.Errorf("BuildStamp() = %s; want = %s", got, tc.want)
        }
    }
}
```

Solution 5: refactoring for testability

ex5/stamp.go

```
var (
    osHostname = os.Hostname
    timeNow    = time.Now
)

// BuildStamp returns a unique build stamp string.
func BuildStamp() string {
    hostname, err := osHostname()
    if err != nil {
        hostname = "<unknown>"
    }
    now := timeNow()
    return fmt.Sprintf("%s@%s", hostname, now.Format(timeFormat))
}
```

Solution 5: stubbing out package functions

ex5/stamp_test.go

```
func TestBuildStamp(t *testing.T) {
    defer func() { timeNow, osHostname = time.Now, os.Hostname }()
    for _, tc := range []struct {
        now      string
        hostname string
        err      error
        want     string
    }{
        {"2015-10-25T10:11:12", "foo", nil, "foo@2015-10-25T10:11:12"},
        {"2015-10-25T10:11:12", "", errors.New("no name"), "<unknown>@2015-10-25T10:11:12"},
    } {
        timeNow = func() time.Time { t, _ := time.Parse(timeFormat, tc.now); return t }
        osHostname = func() (string, error) { return tc.hostname, tc.err }
        if got := BuildStamp(); tc.want != got {
            t.Errorf("BuildStamp() = %s; want = %s", got, tc.want)
        }
    }
}
```

To be continued

'gRPC Testing in Go'

Thank you

Ákos Frohner
szamcsi@google.com (mailto:szamcsi@google.com)