

Go testing: from basic to advanced

2019 November 1

Anderson Queiroz
Blacklane

GolangPiter 2019



TECHNICAL CONFERENCE FOR HARDCORE GO DEVELOPERS

SAINT PETERSBURG
2019 NOVEMBER 1



Go testing: from basic to advanced

Anderson Queiroz
Lead Backend Engineer
Blacklane

 **@ainsoph**

 **in/andersonq**



What is a test in go?

- any file ending in `_test.go`
- any function like `func TestXxx(*testing.T)`

The first character after **Test** MUST be a capital letter

```
func TestSample(t *testing.T) {  
    // your test  
}
```

Simple test

```
func TestSimple(t *testing.T) {  
    expected := 2  
    actual := 3  
    if actual != expected {  
        t.Errorf("expected: %d, actual: %d", expected, actual)  
    }  
}
```

***testing.T?**

*testing.T?

In a test we interact with the type **testing.T**. It provides all functionalities we need to mark a test as failed, skip test and so on.

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

Failing a test

- flag a failure and move on

```
Fail()
```

- mark the who test as failed and stops execution

```
FailNow()
```

- like `Fail()`, but adds a log message (the most used)

```
Error(args ...interface{})  
Errorf(format string, args ...interface{})
```

- like `FailNow()`, but adds a log message

```
Fatal(args ...interface{})  
Fatalf(format string, args ...interface{})
```


(more) Simple tests

```
func TestSimple(t *testing.T) {
    expected := 2
    actual := 3
    if actual != expected {
        t.Errorf("expected: %d, actual: %d", expected, actual)
    }
}
```

```
func TestSimpleFatal(t *testing.T) {
    someSetup := errors.New("setup failed")
    expected, actual := 3, 2

    if someSetup != nil {
        t.Fatalf("set up failed, aborting test: %v", someSetup)
    }
    // Not executed
    if actual != expected {
        t.Errorf("expected: %d, actual: %d", expected, actual)
    }
}
```

Running tests

Running tests

- all tests in a package

```
go test [Package path]
```

- all tests in current folder and sub folders:

```
go test ./...
```

- select test by name/regex

```
go test -run Foo      # Run top-level tests matching "Foo", such as "TestFooBar".  
go test -run Foo/A=  # For top-level tests matching "Foo", run subtests matching "A=".  
go test -run /A=1    # For all top-level tests, run subtests matching "A=1".
```

Verbose mode

```
go test -v
```

- go test will print individually every test
- print the output of **t.Log** function families
- check in runtime by calling **testing.Verbose()**

```
func TestVerbose(t *testing.T) {  
    t.Log("only printed in verbose mode")  
    log.Println("log.Println: always printed")  
  
    if testing.Verbose() {  
        log.Println("some vrebose, but really useful info")  
    }  
}
```

Verbose mode (output)

- not verbose

```
$ go test -run TestVerbose
2019/09/06 12:01:39 log.Println: always printed
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.006s
```

- verbose

```
$ go test -v -run TestVerbose
=== RUN   TestVerbose
2019/09/06 12:02:16 log.Println: always printed
--- PASS: TestVerbose (0.00s)
    tips_test.go:39: only printed in verbose mode
    tips_test.go:43: some verbose, but really useful info
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.007s
```

Short mode

```
go test -short
```

- used to skip *slow tests*
- check in runtime by calling `testing.Short()`

```
func TestShort(t *testing.T) {  
    // Can't be quicker  
}  
  
func TestShortNotSoShort(t *testing.T) {  
    if testing.Short() {  
        t.Skip("Skip in short mode") // needs verbose to show this message  
    }  
    time.Sleep(3 * time.Second)  
}
```

Short mode (output)

- not verbose

```
$ go test -run TestShort
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    3.014s
```

- verbose

```
$ go test -v -short -run TestShort
=== RUN   TestShort
--- PASS: TestShort (0.00s)
=== RUN   TestShortNotSoShort
--- SKIP: TestShortNotSoShort (0.00s)
      tips_test.go:56: Skip in short mode
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.005s
```

Helper functions

Helper functions

Sometimes we need to do some setup. In order to not clog the test, a helper function is used:

```
func TestWithHelper(t *testing.T) {
    helperFunction(t, "a parameter")
    // ...
}

func helperFunction(t *testing.T, param string) string {
    t.Fatal("helperFunction setup failed")
    return param
}
```

Passing `t` is a common practice so if anything fails in the setup we can already abort the test.

Helper functions

```
$ go test -run TestWithHelper
--- FAIL: TestWithHelper (0.00s)
    tips_test.go:97: helperFunction setup failed
FAIL
exit status 1
FAIL    github.com/AndersonQ/golangpiter2019gotesting    0.005s
```

However, the failure output shows the failure was on

```
t.Fatal("helperFunction setup failed")
```

Not so helpful.

t.Helper() for the rescue

t.Helper()

We can invoke **t.Helper()** to signal it's a helper function and the error should be reported on the caller

```
func TestWithBetterHelper(t *testing.T) {
    betterHelperFunction(t, "file.json")
}

func betterHelperFunction(t *testing.T, fileName string) string {
    t.Helper()
    t.Fatal("betterHelperFunction setup failed")
    return ""
}
```

```
$ go test -run TestWithBetterHelper
--- FAIL: TestWithBetterHelper (0.00s)
    tips_test.go:106: betterHelperFunction setup failed
FAIL
exit status 1
FAIL    github.com/AndersonQ/golangpiter2019gotesting    0.005s
```

Table test

Table test

When we have similar test setup, only some values changing from one test to another, we can use a table test

Table test

Lets write a wee test for `math.Abs`

```
func TestTableSimple(t *testing.T) {
    tcs := []struct {
        name string
        val float64
        want float64
    }{
        {name: "the positive", val: 42, want: 42},
        {name: "the negative", val: -42, want: 42},
        {name: "zero", val: 0, want: 0},
    }
    for _, tc := range tcs {
        t.Run(tc.name, func(t *testing.T) {
            got := math.Abs(tc.val)
            if tc.want != got {
                t.Errorf("want: %f, got: %f", tc.want, got)
            }
        })
    }
}
```

Table test (output)

- not verbose:

```
$ go test -run TestTableSimple
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.004s
```

- verbose:

```
$ go test -v -run TestTableSimple
=== RUN   TestTableSimple
=== RUN   TestTableSimple/the_positive
=== RUN   TestTableSimple/the_negative
=== RUN   TestTableSimple/zero
--- PASS: TestTableSimple (0.00s)
    --- PASS: TestTableSimple/the_positive (0.00s)
    --- PASS: TestTableSimple/the_negative (0.00s)
    --- PASS: TestTableSimple/zero (0.00s)
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.004s
```


Slow table test?

```
func TestTableSlow(t *testing.T) {
    tcs := []struct {
        name string
        sleep time.Duration
    }{
        {name: "1s", sleep: 1 * time.Second},
        {name: "2s", sleep: 2 * time.Second},
        {name: "3s", sleep: 3 * time.Second},
    }
    for _, tc := range tcs {
        t.Run(tc.name, func(t *testing.T) {
            time.Sleep(tc.sleep)
        })
    }
}
```

Slow table test? (output)

```
$ go test -v -run TestTableSlow
=== RUN   TestTableSlow
=== RUN   TestTableSlow/1s
=== RUN   TestTableSlow/2s
=== RUN   TestTableSlow/3s
--- PASS: TestTableSlow (6.01s)
    --- PASS: TestTableSlow/1s (1.00s)
    --- PASS: TestTableSlow/2s (2.00s)
    --- PASS: TestTableSlow/3s (3.00s)
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    6.012s
```

Too slow, we can do better!

t.Parallel() for the rescue

t.Parallel()

All tests calling **t.Parallel()** will run in parallel within the same package

```
func TestTableParallel(t *testing.T) {
    tcs := []struct {
        name string
        sleep time.Duration
    }{
        {name: "1s", sleep: 1 * time.Second},
        {name: "2s", sleep: 2 * time.Second},
        {name: "3s", sleep: 3 * time.Second},
    }
    for _, tc := range tcs {
        t.Run(tc.name, func(t *testing.T) {
            t.Parallel()
            t.Log("running ", tc.name)
            time.Sleep(tc.sleep)
        })
    }
}
```

t.Parallel()

```
$ go test -v -run TestTableParallel$
=== RUN   TestTableParallel
=== RUN   TestTableParallel/1s
=== PAUSE TestTableParallel/1s
=== RUN   TestTableParallel/2s
=== PAUSE TestTableParallel/2s
=== RUN   TestTableParallel/3s
=== PAUSE TestTableParallel/3s
=== CONT  TestTableParallel/1s
=== CONT  TestTableParallel/3s
=== CONT  TestTableParallel/2s
--- PASS: TestTableParallel (0.00s)
    --- PASS: TestTableParallel/2s (3.00s)
        tips_test.go:174: running 3s
    --- PASS: TestTableParallel/1s (3.00s)
        tips_test.go:174: running 3s
    --- PASS: TestTableParallel/3s (3.00s)
        tips_test.go:174: running 3s
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    3.008s
```

something looks wrong...

why is it only printing "tips_test.go:174: running 3s" ?

t.Parallel()

The goroutines are sharing the variable **tc**!

```
for _, tc := range tcs {
    t.Run(tc.name, func(t *testing.T) {
        t.Parallel()
        t.Log("running ", tc.name)
        time.Sleep(tc.sleep)
    })
}
```

t.Parallel()

Each goroutine has to have its own **tc**!

```
func TestTableParallelFixed(t *testing.T) {
    tcs := []struct {
        name string
        sleep time.Duration
    }{
        {name: "1s", sleep: 1 * time.Second},
        {name: "2s", sleep: 2 * time.Second},
        {name: "3s", sleep: 3 * time.Second},
    }
    for _, tc := range tcs {
        t.Run(tc.name, func(t *testing.T) {
            tc := tc // rebidding tc so the goroutines will not share it
            t.Parallel()
            t.Log("running ", tc.name)
            time.Sleep(tc.sleep)
        })
    }
}
```

t.Parallel()

Watch out!

If you write something like

```
tc := tc
```

put a comment explaining why!

We don't want a well intentioned engineer removing this "*useless*" line

t.Parallel()

You can control how many tests are run in parallel using the flag `-p`:

```
go test -p=3 ./...
```

or to force them to run sequentially

```
go test -p=1 ./...
```

What else?

Race detector!

Race detector

As we're in the parallelism topic, since go 1.1 we have a built in race detector. Just pass *-race* and it'll kick in.

```
$ go test -race mypkg // test the package
$ go run -race mysrc.go // compile and run the program
$ go build -race mycmd // build the command
$ go install -race mypkg // install the package
```

(from <https://blog.golang.org/race-detector>)

Race detector

```
$ go test -run TestRace
950.594304ms
1.036182019s
1.704839419s
1.944835828s
2.232832234s
2.78703401s
3.424386815s
3.758950018s
3.944828741s
4.427233451s
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting  5.009s
```

Race detector in action

```
$ go test -race -run TestRace
=====
WARNING: DATA RACE
Read at 0x00c000010038 by goroutine 9:
  github.com/AndersonQ/golangpiter2019gotesting.race.func1()
    /Users/aqueiroz/devel/github.com/AndersonQ/golangpiter2019gotesting/race.go:14 +0x121

Previous write at 0x00c000010038 by goroutine 8:
  github.com/AndersonQ/golangpiter2019gotesting.race()
    /Users/aqueiroz/devel/github.com/AndersonQ/golangpiter2019gotesting/race.go:12 +0x18d
  github.com/AndersonQ/golangpiter2019gotesting.TestRace()
    /Users/aqueiroz/devel/github.com/AndersonQ/golangpiter2019gotesting/race_test.go:6 +0x2f
  testing.tRunner()
    /usr/local/go/src/testing/testing.go:909 +0x199
```

Race detector in action (cont.)

Goroutine 9 (running) created at:

```
time.goFunc()  
  /usr/local/go/src/time/sleep.go:168 +0x51
```

Goroutine 8 (running) created at:

```
testing.(*T).Run()  
  /usr/local/go/src/testing/testing.go:960 +0x651  
testing.runTests.func1()  
  /usr/local/go/src/testing/testing.go:1202 +0xa6  
testing.tRunner()  
  /usr/local/go/src/testing/testing.go:909 +0x199  
testing.runTests()  
  /usr/local/go/src/testing/testing.go:1200 +0x521  
testing.(*M).Run()  
  /usr/local/go/src/testing/testing.go:1117 +0x2ff  
github.com/AndersonQ/golangpiter2019gotesting.TestMain()  
  /Users/aqueiroz/devel/github.com/AndersonQ/golangpiter2019gotesting/tips_test.go:24 +0x62  
main.main()  
  _testmain.go:66 +0x223
```

=====

Race detector in action (cont.)

```
--- FAIL: TestRace (5.00s)
    testing.go:853: race detected during execution of test
FAIL
exit status 1
FAIL    github.com/AndersonQ/golangpiter2019gotesting    5.015s
```


testdata

testdata

testdata is a special directory reserved for, well, test data:

"The go tool will ignore a directory named "testdata", making it available to hold ancillary data needed by the tests." (go help test)

Also when running tests the root directory is set to the package root.

```
$ ll
drwxr-xr-x  3 aqueiroz  aqueiroz   96B  6 Sep 14:40 testdata
-rw-r--r--  1 aqueiroz  aqueiroz  4.3K 29 Oct 11:16 tips_test.go
```

testdata

To access *testdata* is as easy as:

```
func TestTestdata(t *testing.T) {
    expected := "Hello, golpher!"

    path := filepath.Join("testdata", "hello")
    file, _ := ioutil.ReadFile(path)

    str := string(file)
    if str != expected {
        t.Fatalf("expected: %s, actual: %s", expected, str)
    }
}
```

Black box testing

"_test" package

"Test files that declare a package with the suffix "_test" will be compiled as a separate package, and then linked and run with the main test binary." (go help test)

- `mypackage_test` can live in the same folder as *mypackage*
- it'll only have access to exported things
- `go test` runs the tests within *mypackage_test* as well as the other tests

```
package pwd_test

import (
    "testing"

    "github.com/AndersonQ/golangpiter2019gotesting/pwd"
)

func TestPackage_test(t *testing.T) {
    pwd.PWD()
}
```

Integration test

Integration test



Dave Cheney

@davecheney

Follow

A test is not a unit test if:

- It talks to the database
- It communicates across the network
- It touches the file system
- It can't run at the same time as any of your other unit tests
- You have to do special things to your environment to run it.

— [@mfeathers](#)

5:46 am - 29 Sep 2019

1,003 Retweets 3,089 Likes

Rb



54

1.0K

3.1K

Integration test

Integration tests or any other tests with external dependencies should be separated from unit tests

We can use the **-short** flag and don't run them when it's set. However we might need it to separate **slow unit tests** or something else

We can use build flags instead! Also naming the test files *_integration_test.go*

Integration test

```
// +build integration

package main

import "testing"

func TestIntegrationTest(t *testing.T) {
    t.Log("A integration test")
}
```

```
$ go test -v -run TestIntegrationTest
testing: warning: no tests to run
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting    0.005s
```

testing: warning: no tests to run!

Integration test

Let's try again passing your build tag **integration**

```
$ go test -v -tags integration -run TestIntegrationTest
=== RUN   TestIntegrationTest
--- PASS: TestIntegrationTest (0.00s)
    tips_integration_test.go:11: A integration test
PASS
ok      github.com/AndersonQ/golangpiter2019gotesting 0.005s
```

Set up and Teardown

func TestMain(m *testing.M)

We can have a *"main"* function for tests which will be called by *go test* and should fire the tests

```
func TestMain(m *testing.M) {  
    // call flag.Parse() here if TestMain uses flags  
  
    globalSetup()  
    exitCode := m.Run()  
    globalTeardown()  
  
    os.Exit(exitCode)  
}
```

Watch out *MainTest* receives a **testing.M* instead of **testing.T*

flags for TestMain

"If TestMain depends on command-line flags, including those of the testing package, it should call flag.Parse explicitly" (<https://golang.org/doc/testing>)

```
func TestMain(m *testing.M) {  
    // call flag.Parse() here if TestMain uses flags  
  
    // If TestMain depends on command-line flags, including those of the testing package,  
    // it should call flag.Parse explicitly (https://golang.org/doc/testing)  
    flag.Parse()  
  
    if testing.Verbose() {  
        fmt.Println("TestMain")  
    }  
  
    globalSetup()  
    exitStatus := m.Run()  
    globalTeardown()  
  
    os.Exit(exitStatus)  
}
```

flags for TestMain

```
$ go test -v ./setupteardown
TestMain
Package level SetUp
=== RUN   TestSetUpTearDown1
--- PASS: TestSetUpTearDown1 (0.00s)
    set_up_tear_down_test.go:18: SetUp/TearDown test 1
=== RUN   TestSetUpTearDown2
--- PASS: TestSetUpTearDown2 (0.00s)
    set_up_tear_down_test.go:22: SetUp/TearDown test 2
    set_up_tear_down_test.go:23: SetUp/TearDown test 2
PASS
Package level TearDown
ok      github.com/AndersonQ/golangpiter2019gotesting/setupteardown    0.005s
```

flags for TestMain broken

```
func TestMain(m *testing.M) {  
    if testing.Verbose() {  
        fmt.Println("TestMain")  
    }  
  
    os.Exit(m.Run())  
}
```

```
$ go test ./setupteardownbroken  
panic: testing: Verbose called before Parse  
  
goroutine 1 [running]:  
testing.Verbose(...)   
    /usr/local/go/src/testing/testing.go:392  
github.com/AndersonQ/golangpiter2019gotesting/setupteardownbroken.TestMain(0xc000b0000)  
    /Users/aqueiroz/devel/github.com/AndersonQ/golangpiter2019gotesting/setupteardownbroken/set_up_  
main.main()  
    _testmain.go:40 +0x135  
FAIL    github.com/AndersonQ/golangpiter2019gotesting/setupteardownbroken    0.007s  
FAIL
```


Test cache

Cached tests

Go will cache tests whenever possible:

"[...] go test caches successful package test results to avoid unnecessary repeated running of tests. When the result of a test can be recovered from the cache, go test will redisplay the previous output instead of running the test binary again. When this happens, go test prints '(cached)' in place of the elapsed time in the summary line." (go help test)

Cached tests

- "The idiomatic way to disable test caching explicitly is to use `-count=1`" (`go help test`)

```
$ go test ./pwd
ok      github.com/AndersonQ/golangpiter2019gotesting/pwd    0.012s

$ go test ./pwd
ok      github.com/AndersonQ/golangpiter2019gotesting/pwd    (cached)

$ go test -count=1 ./pwd
ok      github.com/AndersonQ/golangpiter2019gotesting/pwd    0.012s
```

Questions?

Thank you

Anderson Queiroz
Blacklane

[@AinSoph](http://twitter.com/AinSoph) (<http://twitter.com/AinSoph>)

contato@andersonq.eti.br (<mailto:contato@andersonq.eti.br>)

<https://www.linkedin.com/in/andersonq/> (<https://www.linkedin.com/in/andersonq/>)

<https://github.com/AndersonQ/golangpiter2019gotesting> ([https://github.com/AndersonQ](https://github.com/AndersonQ/golangpiter2019gotesting)

[/golangpiter2019gotesting](https://github.com/AndersonQ/golangpiter2019gotesting))