Go testing style guide

Martin Tournoij

A small (opinionated) style guide for writing Go tests. There is much more to be said about writing good tests than what I've written here. Most of this focuses on *style*, rather than *technique*.

Also see Testing isn't everything.

Use table-drive tests, and consistently use `tt` for a test case

Try to use table-driven tests whenever feasible, but it's okay to just copy some code when it's not; don't force it (e.g. sometimes it's easier to write a table-driven test for all but one or two cases; be practical).

Consistently using the same variable name for a test case will make it easier to work with large code bases. You don't *have* to use `tt`, but it is the most commonly used in Go's standard library (564 times, vs. 116 for `tc`).

Also see TableDrivenTests.

Example:

```
tests := []struct {
    // ...
}{}

for _, tt := range tests {
}
```

Use subtests

Using subtests makes it possible to run just a single test case from a table, as well as easily see which test *exactly* failed. Since subtests are comparatively new (Go 1.7, Oct 2016) many existing tests don't use them, but they should be used for all new tests.

I tend to simply use the test number if it's obvious what is being tested, and add a test name if it's not or if there are many test cases.

Also see Using Subtests and Sub-benchmarks.

Example:

```
tests := []struct {
    // ...
}{}

for i, tt := range tests {
    t.Run(fmt.Sprintf("%d", i), func(t *testing.T) {
        got := TestFunction(tt.input)
        if got != tt.want {
            t.Errorf("failed for %v ...", tt.input)
        }
    })
}
```

Don’t ignore errors

I frequently see people ignore errors in tests. This is a bad idea and can make for some confusing test failures.

Example:

```
got, err := Fun()
if err != nil {
    t.Fatalf("unexpected error: %v", err)
}
```

or:

```
got, err := Fun()
if err != tt.wantErr {
    t.Fatalf("wrong error\ngot:  %v\nwant: %v", err, tt.wantErr)
}
```

I often use `ErrorContains`, which is a useful helper function for testing error messages (avoids some `if err != nil && [..]`).

Lint your tests as your regular code

Tests are code that can fail, be wrong, and will need to be maintained. So if you think it’s worth running a linter on your regular code, then it’s almost certainly also worth running it on your test code (e.g. `go vet`, `errcheck`, etc.)

Use `want` and `got`

`want` is shorter than `expected`, `got` is shorter than `actual`. Shorter names is always good, IMHO, and is especially beneficial for aligning output (see next item).

Example:

```
cases := []struct {
    want     string
    wantCode int
}{}
```

Add useful, aligned, information

It’s annoying when a test fail with a useless error message, or a noisy error message which makes it hard to see what exactly went wrong.

This is not especially useful:

```
t.Errorf("wrong output: %v", got)
```

When it fails it tells us we got the wrong output, but what did we even expect to have?

This is better:

```
name := "test foo"
got := "this string!"
want := "this string"
t.Errorf("wrong output for %v, want %v; got %v", name, got, want)
```

With the downside that it’s hard to see what exactly failed:

```
--- FAIL: TestX (0.00s)
        a_test.go:9: wrong output for test foo, want this string!; got
this string
```

When aligned, this is a lot easier:

```
name := "test foo"
want := "this string"
t.Run(name, func(t *testing.T) {
    got := "this string!"
    t.Errorf("wrong output\ngot:  %q\nwant: %q", got, want)
})
```

```
--- FAIL: TestX (0.00s)
    --- FAIL: TestX/test_foo (0.00s)
        a_test.go:10: wrong output
```

```
                got:  "this string!"
                want: "this string"
```

Notice the two spaces after `got:` to make it aligned with `want`. If I had used `expected` I would have to use six spaces.

I also tend to prefer to use `%q` or `%#v`, as that will show things like trailing whitespace or unprintable characters more clearly.

Use a diff when comparing larger objects; for example with go-cmp:

```
if d := cmp.Diff(got, tt.want); d != "" {
    t.Errorf("(-got +want)\n:%s", d)
}
```

```
--- FAIL: TestParseFilter (0.00s)
    --- FAIL: TestParseFilter/alias (0.00s)
        query_test.go:717: (-got +want)
            :{jsonapi.Filter}.Alias:
                -: "fail"
                +: "alias"
```

Make it clear what is being tested

Sometimes I see tests where I wonder "what is this even testing?" This can be especially confusing if the test fails for unclear reasons. What should be fixed? Is the test even correct?

Example:

```
cases := []struct {
    name string
}{
    {
        "space after @",
    },
    {
        "unicode space before @",
    },
    // ...
}
```

If adding `name` to existing test cases is too much work then comment can be fine too.