Add a test - The Go Programming Language

Read time: 3 minutes

---

1. Documentation
2. Tutorials
3. Add a test

Now that you've gotten your code to a stable place (nicely done, by the way), add a test. Testing your code during development can expose bugs that find their way in as you make changes. In this topic, you add a test for the `Hello` function.

Go's built-in support for unit testing makes it easier to test as you go. Specifically, using naming

conventions, Go's `testing` package, and the `go test` command, you can quickly write and execute tests.

1. In the greetings directory, create a file called greetings_test.go.

   Ending a file's name with _test.go tells the `go test` command that this file contains test functions.

2. In greetings_test.go, paste the following code and save the file.

```
package greetings

import (
    "testing"
    "regexp"
)

// TestHelloName calls greetings.Hello with a name, checking
// for a valid return value.
func TestHelloName(t *testing.T) {
    name := "Gladys"
    want := regexp.MustCompile(`\b`+name+`\b`)
    msg, err := Hello("Gladys")
    if !want.MatchString(msg) || err != nil {
        t.Fatalf(`Hello("Gladys") = %q, %v, want match for %#q,
nil`, msg, err, want)
    }
```

```
}

// TestHelloEmpty calls greetings.Hello with an empty string,
// checking for an error.
func TestHelloEmpty(t *testing.T) {
    msg, err := Hello("")
    if msg != "" || err == nil {
        t.Fatalf(`Hello("") = %q, %v, want "", error`, msg, err)
    }
}
```

In this code, you:

- Implement test functions in the same package as the code you're testing.
- Create two test functions to test the `greetings.Hello` function. Test function names have the form `Test`*`Name`*, where *Name* says something about the specific test. Also, test functions take a pointer to the `testing` package's `testing.T` type as a parameter. You use this parameter's methods for reporting and logging from your test.
- Implement two tests:

- `TestHelloName` calls the `Hello` function, passing a `name` value with which the function should be able to return a valid response message. If the call returns an error or an unexpected response message (one that doesn't include the name you passed in), you use the `t` parameter's `Fatalf` method to print a message to the console and end execution.
- `TestHelloEmpty` calls the `Hello` function with an empty string. This test is designed to confirm that your error handling works. If the call returns a non-empty string or no error, you use the `t` parameter's `Fatalf` method to print a message to the console and end execution.

3. At the command line in the greetings directory, run the `go test` command to execute the test.

The `go test` command executes test functions (whose names begin with `Test`) in test files (whose names end with _test.go). You can add the `-v` flag to get verbose output that lists all of the tests and their results.

The tests should pass.

```
$ go test
PASS
ok      example.com/greetings   0.364s

$ go test -v
=== RUN   TestHelloName
--- PASS: TestHelloName (0.00s)
=== RUN   TestHelloEmpty
--- PASS: TestHelloEmpty (0.00s)
PASS
ok      example.com/greetings   0.372s
```

4. Break the `greetings.Hello` function to view a failing test.

   The `TestHelloName` test function checks the return value for the name you specified as a `Hello` function parameter. To view a failing test result,

change the `greetings.Hello` function so that it no longer includes the name.

In greetings/greetings.go, paste the following code in place of the `Hello` function. Note that the highlighted lines change the value that the function returns, as if the `name` argument had been accidentally removed.

```
// Hello returns a greeting for the named person.
func Hello(name string) (string, error) {
    // If no name was given, return an error with a message.
    if name == "" {
        return name, errors.New("empty name")
    }
    // Create a message using a random format.
    // message := fmt.Sprintf(randomFormat(), name)
    message := fmt.Sprint(randomFormat())
    return message, nil
}
```

5. At the command line in the greetings directory, run `go test` to execute the test.

   This time, run `go test` without the `-v` flag. The output will include results for only the tests that

failed, which can be useful when you have a lot of tests. The `TestHelloName` test should fail -- `TestHelloEmpty` still passes.

```
$ go test
--- FAIL: TestHelloName (0.00s)
    greetings_test.go:15: Hello("Gladys") = "Hail, %v! Well
met!", <nil>, want match for `\bGladys\b`, nil
FAIL
exit status 1
FAIL    example.com/greetings   0.182s
```

In the next (and last) topic, you'll see how to compile and install your code to run it locally.

< Return greetings for multiple people Compile and install the application >