Testing Go Projects

About the trainer

bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

DISCUSSION

What’s your main learning objective?

Why Testing?

On the Importance of Testing

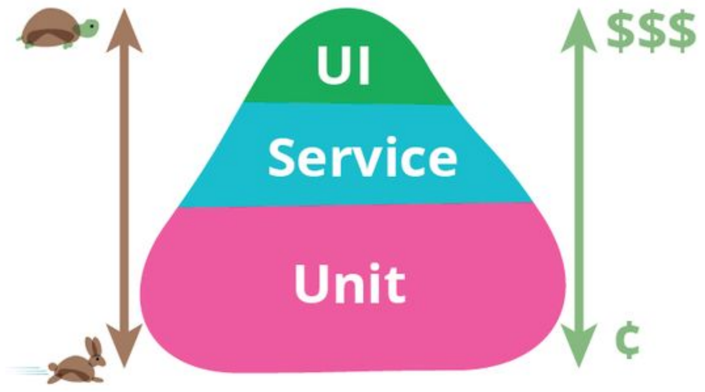
On the Importance of Testing

Delivering a product with acceptable quality

- Customer requirements have been fulfilled
- Ensure the quality of the software, avoiding bugs
- Lower future maintenance cost
- Increase the speed of “time to market”

The Testing Pyramid

Which types of tests are we going to cover today?



https://martinfowler.com/bliki/TestPyramid.html

Testing Basics

Writing and executing tests based on conventions and best practices

The Standard `testing` Package

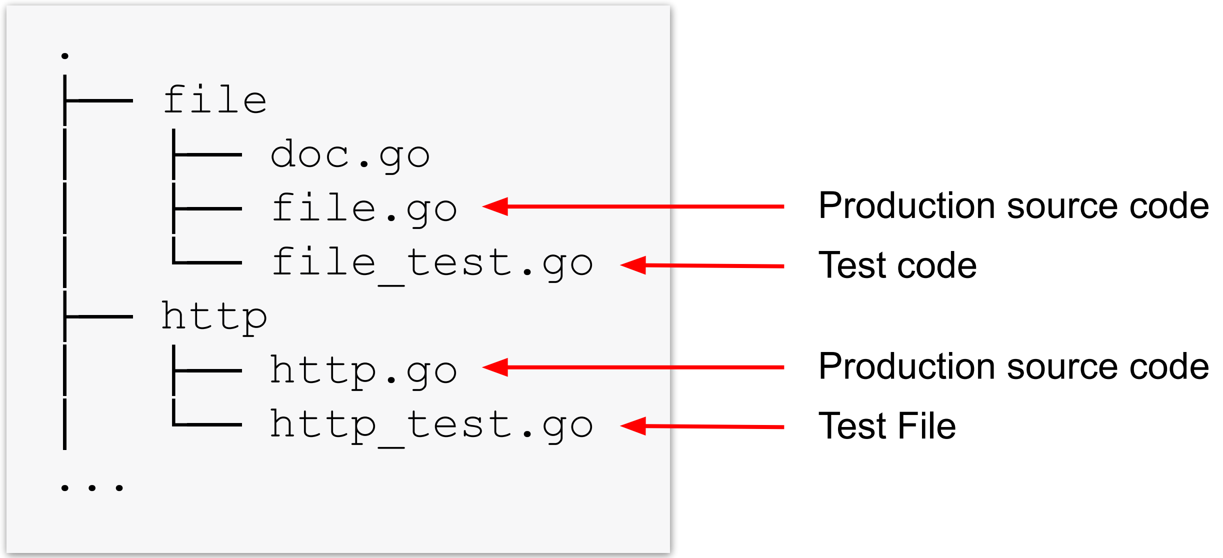
Testing is a first-class citizen in Go

- Built-in feature of Go library
- Easy to understand, no syntactic sugar
- No convenient assertion statements

https://golang.org/pkg/testing/

Test File Conventions

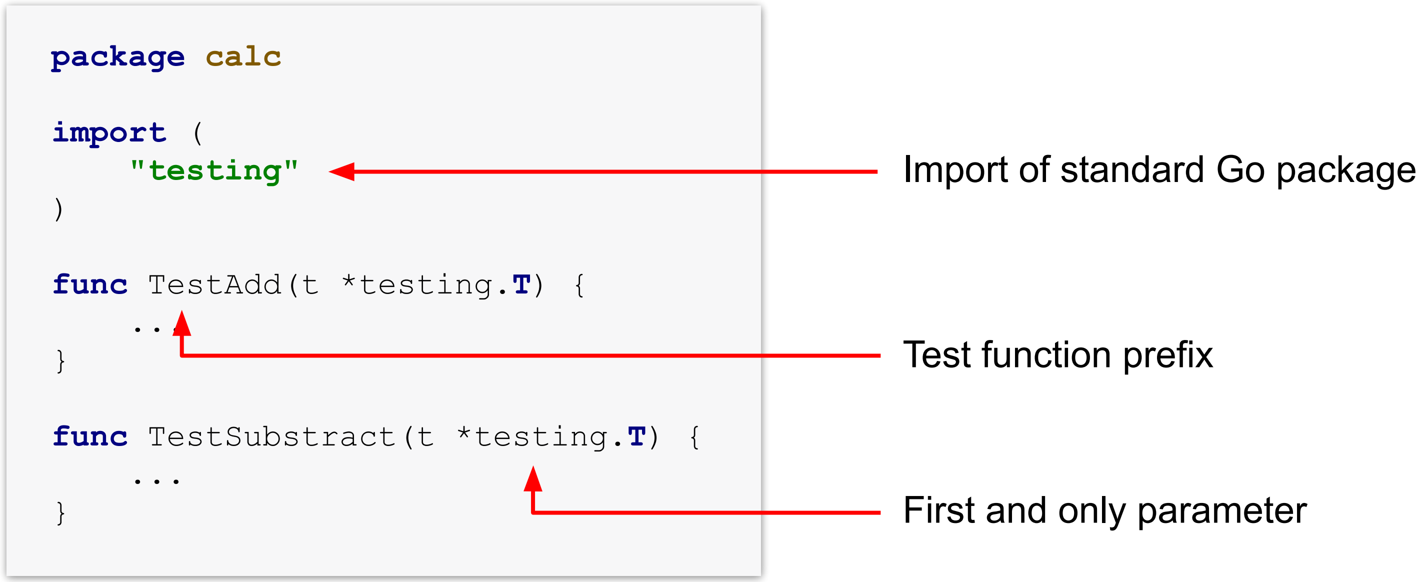
Follows the naming pattern `*_test.go`



```
.
├── file
│   ├── doc.go
│   ├── file.go
│   └── file_test.go
├── http
│   ├── http.go
│   └── http_test.go
...
```

Test Function Conventions

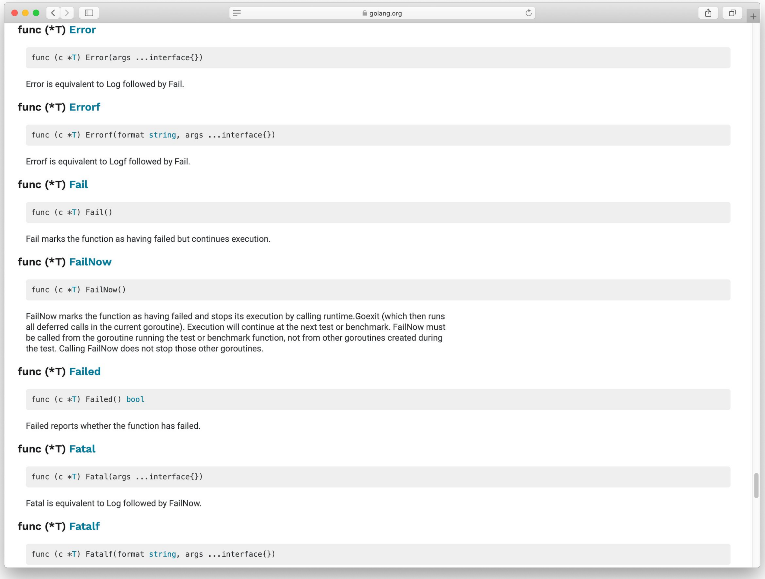
Exported function prefixed with `Test`, single param



```go
package calc

import (
    "testing"
)

func TestAdd(t *testing.T) {
    ...
}

func TestSubstract(t *testing.T) {
    ...
}
```

Verifying the Outcome

Various functions available in `testing` *API*


golang.org

func (*T) Error

```
func (c *T) Error(args ...interface{})
```

Error is equivalent to Log followed by Fail.

func (*T) Errorf

```
func (c *T) Errorf(format string, args ...interface{})
```

Errorf is equivalent to Logf followed by Fail.

func (*T) Fail

```
func (c *T) Fail()
```

Fail marks the function as having failed but continues execution.

func (*T) FailNow

```
func (c *T) FailNow()
```

FailNow marks the function as having failed and stops its execution by calling runtime.Goexit (which then runs all deferred calls in the current goroutine). Execution will continue at the next test or benchmark. FailNow must be called from the goroutine running the test or benchmark function, not from other goroutines created during the test. Calling FailNow does not stop those other goroutines.

func (*T) Failed

```
func (c *T) Failed() bool
```

Failed reports whether the function has failed.

func (*T) Fatal

```
func (c *T) Fatal(args ...interface{})
```

Fatal is equivalent to Log followed by FailNow.

func (*T) Fatalf

```
func (c *T) Fatalf(format string, args ...interface{})
```

https://golang.org/pkg/testing

Writing Meaningful Assertions

Most important part of a test case

```go
func TestAdd(t *testing.T) {
    result := Add(1, 2)

    if result != 3 {
        t.Errorf("Result was incorrect, got: %d, want: %d.", result, 3)
    }
}
```

Code under test

Signaling failure

Failure condition

Failure States

Pick the appropriate state for the right job

`t.Error*` → Report test failures but continue executing the test suite

`t.Fail*` → Report test failures and stop test execution immediately

Test Variable Conventions

Be descriptive, especially with assertion variables

```
func TestAdd(t *testing.T) {
    a := 1
    b := 2
    expected := 3
    actual := Add(a, b)

    if expected != actual {
        t.Errorf("Result was incorrect, got: %d, want: %d.", actual, expected)
    }
}
```

Expected value

Actual value

Easier Assertions with Testify

“Less code, more win” if you are open to external packages

```
import (
     . "github.com/bmuschko/go-testing-frameworks/calc"
     "github.com/stretchr/testify/assert"
     "testing"
)

func TestAdd(t *testing.T) {
     a := 1
     b := 2
     expected := 3
     actual := Add(a, b)

     assert.Equal(t, expected, actual)
}
```

Package import

Assertion usage

Running Tests

Pick one or many packages for executing tests

```
$ go test ./...
?    github.com/bmuschko/letsgopher       [no test files]
ok   github.com/bmuschko/letsgopher/cmd         0.332s
ok   github.com/bmuschko/letsgopher/template/archive     0.341s
ok   github.com/bmuschko/letsgopher/template/config  0.231s
ok   github.com/bmuschko/letsgopher/template/download    0.283s
ok   github.com/bmuschko/letsgopher/template/environment     0.135s
?    github.com/bmuschko/letsgopher/template/prompt  [no test
files]
ok   github.com/bmuschko/letsgopher/template/storage     0.199s
?    github.com/bmuschko/letsgopher/testhelper  [no test files]
```

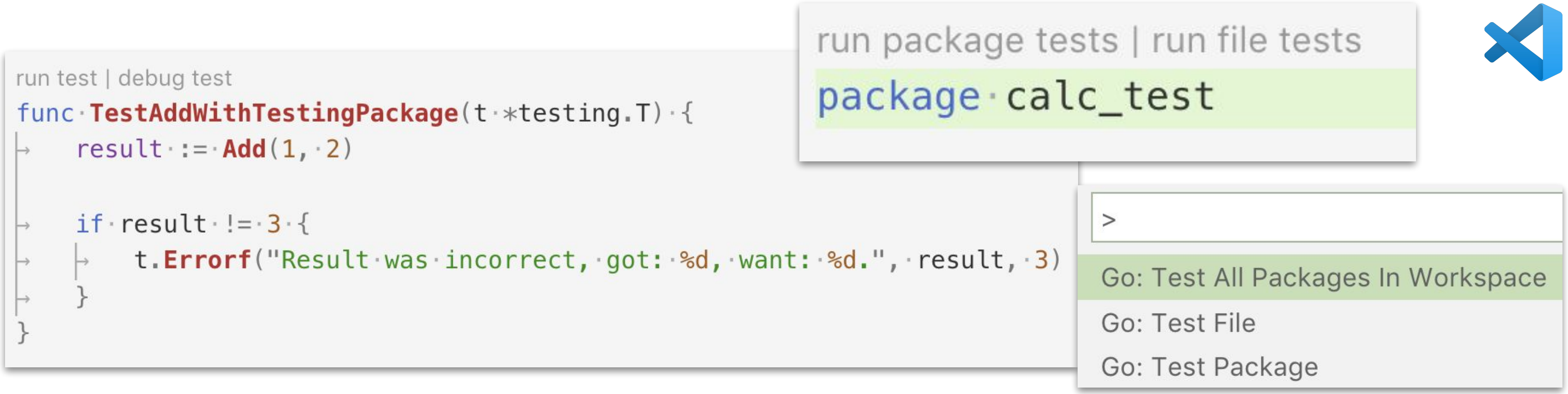
Executing Tests with Details

Breaks down results by test cases, outcome and duration

```
$ go test ./... -v
?    github.com/bmuschko/letsgopher     [no test files]
=== RUN   TestCreateProjectWithoutRegisteredTemplate
--- PASS: TestCreateProjectWithoutRegisteredTemplate (0.00s)
=== RUN   TestCreateProjectWithRegisteredTemplate
--- PASS: TestCreateProjectWithRegisteredTemplate (0.00s)
    create_test.go:66: PASS:  LoadManifestFile(string)
    create_test.go:66: PASS:
Extract(string,string,map[string]interface {})
...
```

IDE integration in VSCode

“Go extension” provides ability to run and debug tests



```
run test | debug test
func TestAddWithTestingPackage(t *testing.T) {
    result := Add(1, 2)

    if result != 3 {
        t.Errorf("Result was incorrect, got: %d, want: %d.", result, 3)
    }
}
```

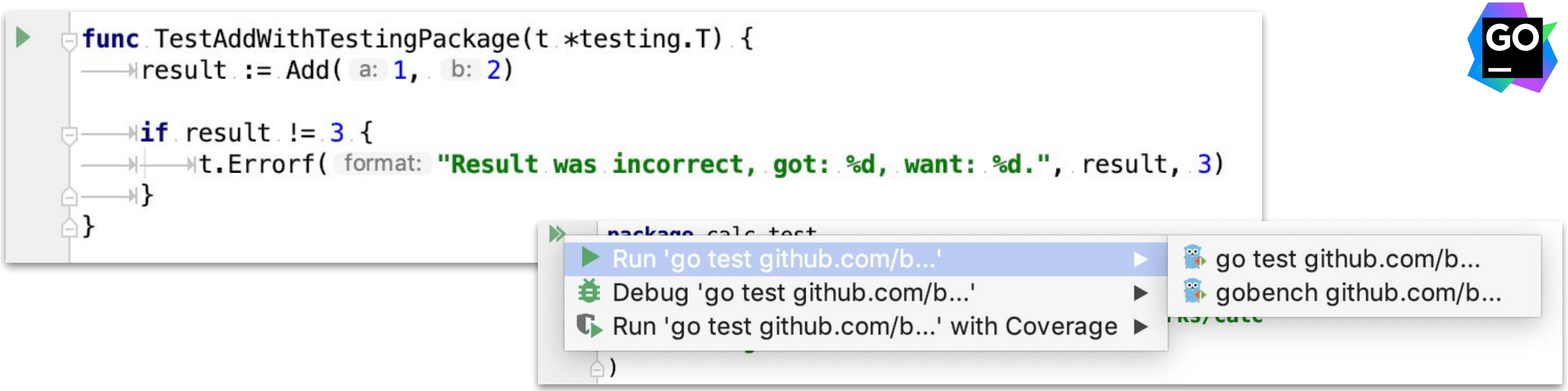
```
run package tests | run file tests
package calc_test
```

```
>
Go: Test All Packages In Workspace
Go: Test File
Go: Test Package
```

https://code.visualstudio.com/docs/languages/go

IDE integration in GoLand

Running and debugging tests, benchmarks and checks



```
func TestAddWithTestingPackage(t *testing.T) {
	result := Add( a: 1,  b: 2)

	if result != 3 {
		t.Errorf( format: "Result was incorrect, got: %d, want: %d.", result, 3)
	}
}
```

package calc_test
Run 'go test github.com/b...'
Debug 'go test github.com/b...'
Run 'go test github.com/b...' with Coverage
go test github.com/b...
gobench github.com/b...

EXERCISE

Implementation and Execution of a Test Case

DISCUSSION

Testing exported vs. internal functions

Good Testing Practices

Avoid writing tests for internal functions

```go
package storage

import "path/filepath"

type Home string

func (h Home) ArchiveDir() string {
	return h.path("archive")
}

func (h Home) path(elem ...string) string {
	p := []string{h.String()}
	p = append(p, elem...)
	return filepath.Join(p...)
}
```

Exported function ✓ **Test**

Internal function ✗ **Don’t Test**

Good Testing Practices

Put test code into different package than code under test

```go
package calc_test

import (
    . "github.com/bmuschko/go-testing-frameworks/calc"
    "testing"
)

func TestAdd(t *testing.T) {
    result := Add(1, 2)
    ...
}
```

Can only access exported functions

Import code under test with “dot” notation

Good Testing Practices

Testing internals indicates a potential need for refactoring

- Demonstrates how the end user would API
- Exposes implementation details not relevant to end users
- After consideration, decide to export API

BREAK

5 mins

Capturing Code Coverage Metrics

"Which portion of your code has been exercised?"

```
$ go test ./... -coverprofile=coverage.txt -covermode=count
?     github.com/bmuschko/letsgopher      [no test files]
ok    github.com/bmuschko/letsgopher/cmd 0.709s    coverage: 72.3% of statements
ok    github.com/bmuschko/letsgopher/template/archive   0.540s    coverage: 69.6% of
statements
ok    github.com/bmuschko/letsgopher/template/config    0.408s    coverage: 94.1% of
statements
ok    github.com/bmuschko/letsgopher/template/download  0.665s    coverage: 78.6% of
statements
...
```

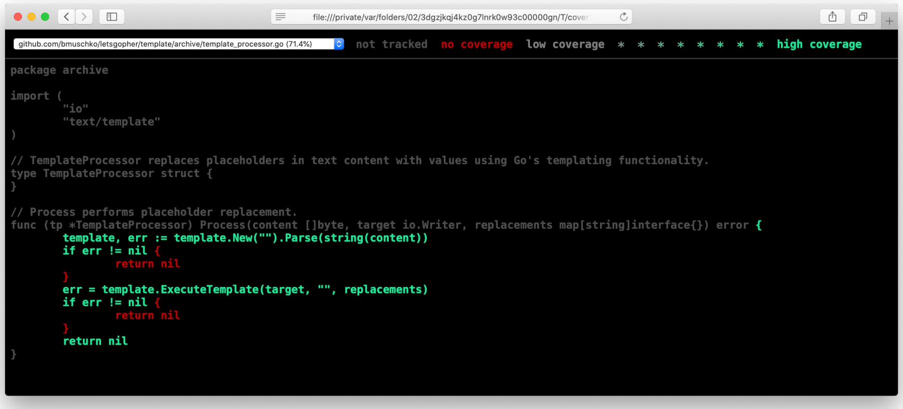
$\xrightarrow{\hspace{3em}}$ `coverage.txt`

Rendering HTML Report

Ad-hoc, browsable coverage visualization

```
$ go tool cover -html=coverage.txt
```

coverage.txt



file:///private/var/folders/02/3dgzjkqj4kz0g7lnrk0w93c00000gn/T/cover

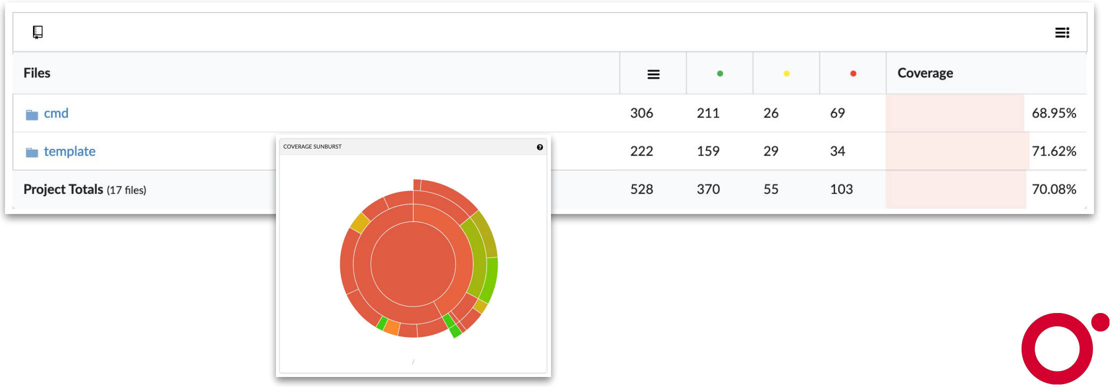
```
github.com/bmuschko/letsgopher/template/archive/template_processor.go (71.4%)   not tracked  no coverage  low coverage  *  *  *  *  *  *  *  *  high coverage

package archive

import (
        "io"
        "text/template"
)

// TemplateProcessor replaces placeholders in text content with values using Go's templating functionality.
type TemplateProcessor struct {
}

// Process performs placeholder replacement.
func (tp *TemplateProcessor) Process(content []byte, target io.Writer, replacements map[string]interface{}) error {
        template, err := template.New("").Parse(string(content))
        if err != nil {
                return nil
        }
        err = template.ExecuteTemplate(target, "", replacements)
        if err != nil {
                return nil
        }
        return nil
}
```

Third-Party Coverage Visuals

Post-process plain-text metrics and report hosting

```
$ bash <(curl -s https://codecov.io/bash)
```

coverage.txt

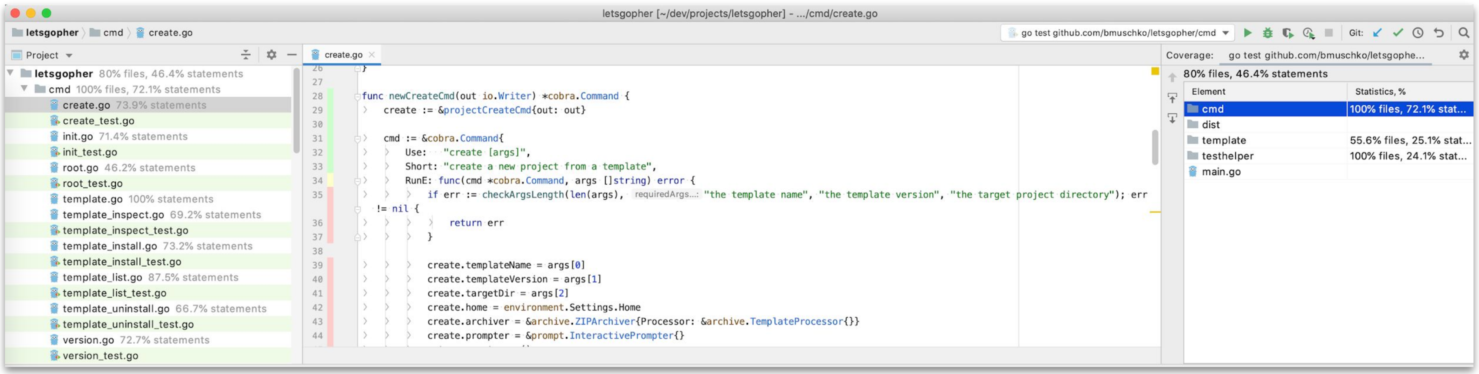


Files	≡	●	●	●	Coverage
cmd	306	211	26	69	68.95%
template	222	159	29	34	71.62%
Project Totals (17 files)	528	370	55	103	70.08%

https://codecov.io/

Coverage in the IDE

Extremely helpful during developments



letsgopher [~/dev/projects/letsgopher] - .../cmd/create.go

letsgopher 〉 cmd 〉 create.go

go test github.com/bmuschko/letsgopher/cmd

Project

- letsgopher 80% files, 46.4% statements
  - cmd 100% files, 72.1% statements
    - create.go 73.9% statements
    - create_test.go
    - init.go 71.4% statements
    - init_test.go
    - root.go 46.2% statements
    - root_test.go
    - template.go 100% statements
    - template_inspect.go 69.2% statements
    - template_inspect_test.go
    - template_install.go 73.2% statements
    - template_install_test.go
    - template_list.go 87.5% statements
    - template_list_test.go
    - template_uninstall.go 66.7% statements
    - template_uninstall_test.go
    - version.go 72.7% statements
    - version_test.go

create.go

```go
}

func newCreateCmd(out io.Writer) *cobra.Command {
	create := &projectCreateCmd{out: out}

	cmd := &cobra.Command{
		Use:   "create [args]",
		Short: "create a new project from a template",
		RunE: func(cmd *cobra.Command, args []string) error {
			if err := checkArgsLength(len(args), requiredArgs...: "the template name", "the template version", "the target project directory"); err != nil {
				return err
			}

			create.templateName = args[0]
			create.templateVersion = args[1]
			create.targetDir = args[2]
			create.home = environment.Settings.Home
			create.archiver = &archive.ZIPArchiver{Processor: &archive.TemplateProcessor{}}
			create.prompter = &prompt.InteractivePrompter{}
```

Coverage: go test github.com/bmuschko/letsgophe...

80% files, 46.4% statements

Element	Statistics, %
cmd	100% files, 72.1% stat...
dist	
template	55.6% files, 25.1% stat...
testhelper	100% files, 24.1% stat...
main.go	

Percentage per package

Color-coded coverage per line

Coverage Overview

EXERCISE

Producing an HTML Report for Code Coverage Metrics

Q & A

5 mins

Advanced Techniques

Navigating day to day challenges

Data Needed as Setup For Test

Test data as code vs. externalized files

```
func TestProcessTemplate(t *testing.T) {
    content := []byte(`{{ if .condition }}
Show this section if the condition is true
{{ else }}
Show this section if the condition is false
{{ end }}`)

    // Replace variable value
    ...
}
```

→

`template.txt`

Where to Store Test Data?

The `go tool` *ignores the directory named* `testdata`

```
$ tree testdata
testdata
└── template
    ├── conditional.txt
    ├── plain_text.txt
    └── replacement.txt
```

```go
func TestProcessTemplate(t *testingT) {
    content := readFile("testdata/template/conditional.txt")

    // Replace variable value
    ...
}
```

Setup & Clean Up Functions

There's no automatism in standard `testing` package

```
func setup(t *testing.T) {
    // Create temporary directory
}

func teardown(t *testing.T) {
    // Delete temporary directory
}

func TestFileProcessing(t *testing.T) {
    setup(t)
    defer teardown(t)

    // Processing files in temp dir
    ...
}
```

Fixture functions

Call explicitly

Built-in Clean Up Function

Doesn’t require a call to `defer`*, called at end of test*

```go
func TestFileProcessing(t *testing.T) {
    // Create temporary directory

    t.Cleanup(func() {
        // Delete temporary directory
    })
}
```

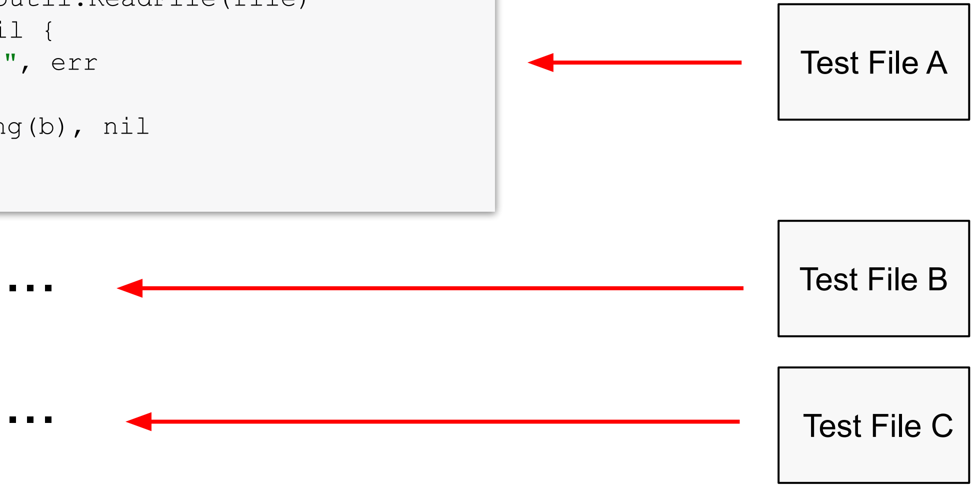
EXERCISE

Setting up and Tearing Down Test Fixtures

Repetitive Test Code

Test infrastructure code is often copy-pasted

```
func ReadFile(file string) (string, error) {
    b, err := ioutil.ReadFile(file)
    if err != nil {
        return "", err
    }
    return string(b), nil
}
```



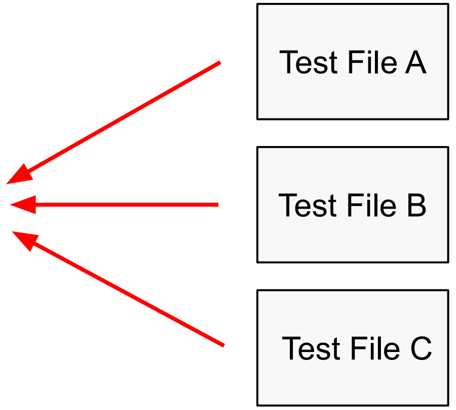
3 copies!

Reusing Test Code

Defined once, reused multiple times

```
func ReadFile(file string) (string, error) {
    b, err := ioutil.ReadFile(file)
    if err != nil {
        return "", err
    }
    return string(b), nil
}
```

1 definition!



Implementing a Test Helper

Defined once, reused multiple times

```
github.com
└── bmuschko
    └── letsgopher
        ├── cmd
        ├── template
        └── testhelper
            └── file_helper.go
```

```go
package cmd

import "github.com/bmuschko/letsgopher/testhelper"

func TestInstallNewTemplate(t *testing.T) {
    templates, e := testhelper.ReadFile(f)
    if e != nil {
        t.Fatalf("failed to read file %s", f)
    }
    ...
}
```

```go
func ReadFile(file string) (string, error) {
    ...
    return string(b), nil
}
```

Error handling
for every use!

Handling Error States in Helper

Don't return `error`, signal error state internally

```go
func ReadFile(t *testing.T, file string) string {
    b, err := ioutil.ReadFile(file)
    if err != nil {
        t.Fatalf("Failed to read file %s. Reason: %s", file, err)
    }
    return string(b)
}
```

```go
package cmd

import "github.com/bmuschko/letsgopher/testhelper"

func TestInstallNewTemplate(t *testing.T) {
    templates := testhelper.ReadFile(t, f)
    ...
}
```

EXERCISE

Implementing a
Test Helper

BREAK

5 mins

Test Case Permutations

Same test logic, but different given/expected data

```go
func TestAddSmallNumbers(t *testing.T) {
    a := 1
    b := 2
    expected := 3
    result := Add(a, b)

    if result != expected {
        t.Errorf("Result was incorrect, " +
                 "got: %d, want: %d.",
                 result, expected)
    }
}
```

```go
func TestAddHighNumbers(t *testing.T) {
    a := 3642
    b := 1834
    expected := 5476
    result := Add(a, b)

    if result != expected {
        t.Errorf("Result was incorrect, " +
                 "got: %d, want: %d.",
                 result, expected)
    }
}
```

Duplication

Table Representation

“I do see similarities! Let’s unify.”

Argument 1	Argument 2	Code Under Test	Expected Result
1	2	`Add(1, 2)`	3
3642	1834	`Add(3642, 1834)`	5476

Opportunity

Table-Driven Tests

Reuse test logic for multiple permutations

```go
func TestAdd(t *testing.T) {
    cases := []struct {
        a          int
        b          int
        expected   int
    }{
        {
            a:          1,
            b:          2,
            expected:   3,
        },
        {
            a:          3642,
            b:          1834,
            expected:   5476,
        },
     }
```

+

```go
for _, c := range cases {
    got := Add(c.a, c.b)

    if got != c.expected {
        t.Errorf("Result was incorrect, " +
                 "got: %d, want: %d.",
                 got, c.expected)
        }
    }
}
```

Data

Test iterations

Test Execution

The output doesn't render the iterations

```
$ go test ./... -v
=== RUN   TestTableAdd
--- PASS: TestTableAdd (0.00s)
PASS
ok    github.com/bmuschko/calc0.101s
```

Successful test result

Suboptimal

```
$ go test ./... -v
=== RUN   TestTableAdd
--- FAIL: TestTableAdd (0.00s)
    calc_test.go:52: Result was incorrect, got: 3, want: 5.
FAIL
FAIL  github.com/bmuschko/calc0.104s
```

Failed test result

Test Execution as Subtest

Iterations run in blocking goroutine

```
for _, c := range cases {
    t.Run(fmt.Sprintf("%d+%d", c.a, c.b), func(t *testing.T) {
        got := Add(c.a, c.b)

        if got != c.expected {
            ...
        }
    }
}
```

Improved output

```
$ go test ./... -v
=== RUN   TestTableAdd
=== RUN   TestTableAdd/1+2
=== RUN   TestTableAdd/3642+1834
--- FAIL: TestTableAdd (0.00s)
    --- FAIL: TestTableAdd/1+2 (0.00s)
        calc_test.go:56: Result was incorrect, got: 3, want: 5.
    --- PASS: TestTableAdd/3642+1834 (0.00s)
FAIL
FAIL    github.com/bmuschko/calc    0.205s
```

Parallel Test Execution

Good idea but beware the gotchas!

```
for _, c := range cases {
    c := c
    t.Run(fmt.Sprintf("%d+%d", c.a, c.b), func(t *testing.T) {
        t.Parallel()
        got := Add(c.a, c.b)

        if got != c.expected {
            ...
        }
    }
}
```

Mandatory:
Capture range variable

Run in parallel

https://github.com/golang/go/wiki/CommonMistakes#using-goroutines-on-loop-iterator-variables

Beware!

EXERCISE

Implementing a
Table-Driven Test

Test Doubles

Emulating real objects for the purpose of testing

Stub

Spy

Fake

Mock

...

https://martinfowler.com/bliki/TestDouble.html

Why and What of Mock Objects

Stand in for complex, real objects

- Mimic the behavior of real objects in controlled ways
- Avoids the need for having real services with expected state
- Sometimes leads to better, more abstracted code
- Mock objects meet the interface requirements

Looking at a Real Object

Interfaces are a necessity for mocking

```go
package download

type Downloader interface {
    Download(url string) (string, error)
}
```

Used as dependency somewhere else

Defining Mock Behavior

Third-party packages help avoid boilerplate code

```go
import "github.com/stretchr/testify/mock"

type DownloaderMock struct {
    mock.Mock
}

func (d *DownloaderMock) Download(url string) (string, error)
{
    args := d.Called(url)
    return args.String(0), args.Error(1)
}
```

Invoking Mock Behavior

Create, inject and assert mock expectations

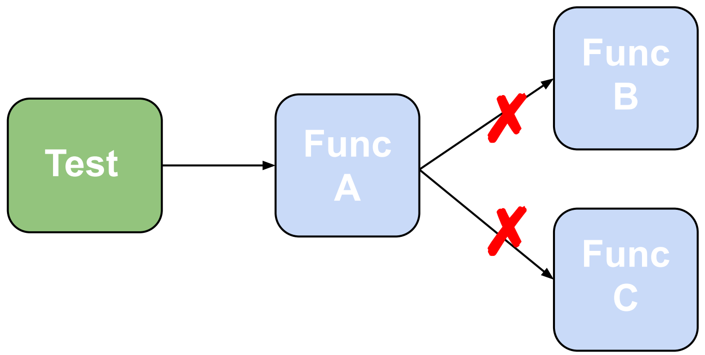
```go
func TestInstall(t *testing.T) {
    dM := new(DownloaderMock)
    dM.On("Download", "http://my.repo.com/hello-world-1.0.0.zip")
      .Return("/my/path/new-project/hello-world-1.0.0.zip", nil)

    // Inject mock object
    // Call code under test

    dM.AssertExpectations(t)
}
```

EXERCISE

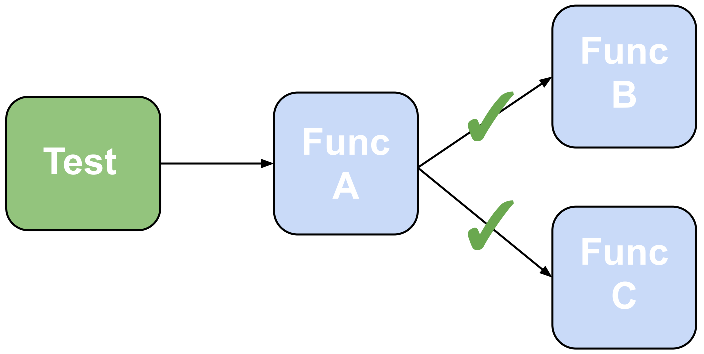
Testing in Isolation by Mocking an Interface

Unit vs. Integration Tests

Unit tests interact with mocks, integ. tests with real services



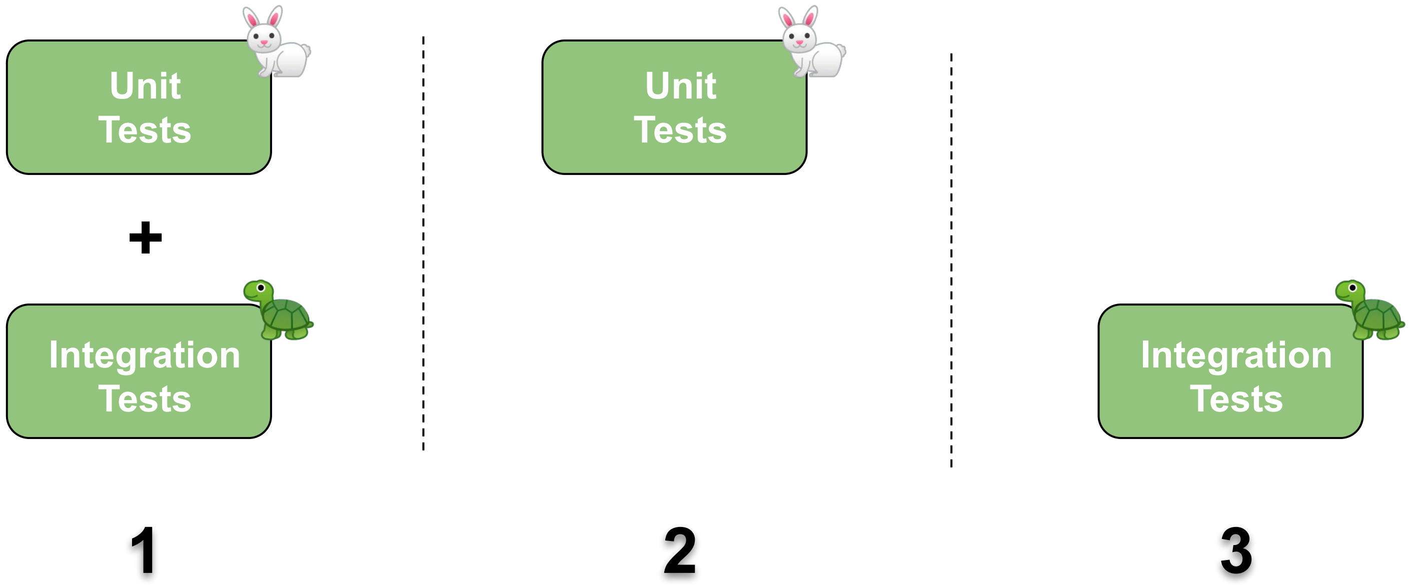
Unit tests



Integration tests

Running Specific Types of Tests

You will likely not want to run all tests all the time



Running All Tests

Simply execute the `test` command as usual

Unit Tests 🐇

+

Integration Tests 🐢

1

```
$ go test ./... -v
=== RUN   TestUnitCreateProjectWithoutRegisteredTemplate
--- PASS: TestUnitCreateProjectWithoutRegisteredTemplate (0.00s)
...
=== RUN   TestIntegrationExtractWithoutTemplateReplacement
--- PASS: TestIntegrationExtractWithoutTemplateReplacement (0.00s)
...
```

Differentiate by Naming Patterns

Clearly indicate type of test by `Test<Suffix>` *or similar*

Unit Tests

```
func TestUnitBuildZipFile(t *testingT) {
    ...
}
```

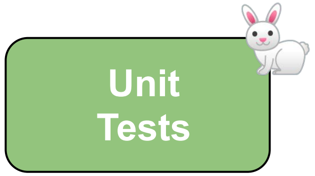
2

Integration Tests

```
func TestIntegrationBuildZipFile(t *testingT) {
    if testing.Short() {
        t.Skip("Skipping integration test")
    }
    ...
}
```

Running Only Unit Tests

Use the built-in command line flag `-short`

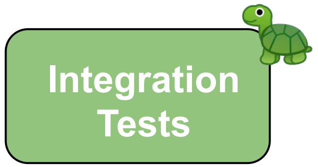


2

```
$ go test ./... -v -short
=== RUN   TestUnitCreateProjectWithoutRegisteredTemplate
--- PASS: TestUnitCreateProjectWithoutRegisteredTemplate (0.00s)
...
=== RUN   TestIntegrationExtractWithoutTemplateReplacement
--- SKIP: TestIntegrationExtractWithoutTemplateReplacement (0.00s)
zip_archiver_test.go:45: template replacements are currently not
working
...
```

Running Only Integration Tests

Execute the `test` command with pattern matching



3

```
$ go test ./... -v -run 'TestIntegration'
ok      github.com/bmuschko/letsgopher/cmd  0.660s [no tests to run]
...
=== RUN   TestIntegrationExtractWithoutTemplateReplacement
--- PASS: TestIntegrationExtractWithoutTemplateReplacement (0.00s)
ok      github.com/bmuschko/letsgopher/template  0.386s [no tests to run]
testing: warning: no tests to run
...
```

Alternative: Build Tags

Requires tagging all test files, sometimes as negated value

```go
// +build !unit
// +build integration

package download

import (
	"testing"
)

...
```

```
$ go test ./... -v -tags=integration
=== RUN   TestIntegrationExtract
--- PASS: TestIntegrationExtract (0.00s)
...
```

EXERCISE

Differentiating Between Unit and Integration Tests

Q & A

5 mins

BREAK

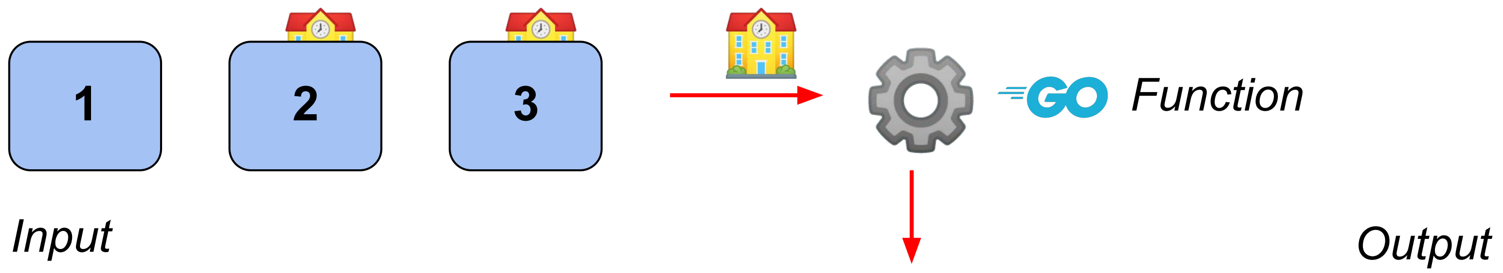
5 mins

Testing Real-World Scenarios

Implementing more complex test cases

Verifying Cumbersome Output

Hard-coding expected values can make tests unreadable



```
| Title          | Trainer        | Duration   | Skill Level
|
| ------------- | ------------- | --------- | -------------
|
| Go             | Ben            | 4 hours    | Beginner
|
| Java Threads   | John           | 8 hours    | Advanced
|
```

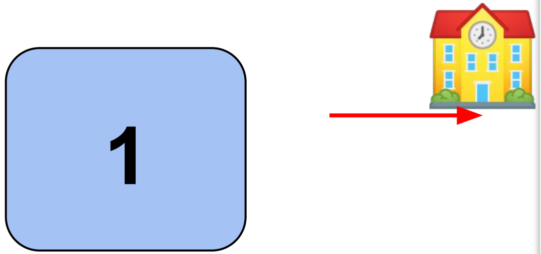
Production Source Code

Hard-coded header, rows derived of data

```
const (
      header string = `
| Title         | Trainer       | Duration  | Skill Level   |
| ------------- | ------------- | --------- | ------------- |
`
      row string = "|  {{ .Title }}  |  {{ .Trainer }}  |  {{ .Duration }}  |  {{ .SkillLevel }}  |\n"
)

func Generate(trainings []Training) string {
    // Using templating to generate table output
}
```

Production Source Code

Data representation implemented as `struct`



```go
type Training struct {
    Title       string
    Trainer     string
    Duration    int
    SkillLevel  string
}
```

Table-Driven Test

Test code can become extremely elaborate and lengthy

```
var trainings []Training = []Training {
    Training {
        Title:        "Go",
        Trainer:      "Ben",
        Duration:     4,
        SkillLevel:  "Beginner",
    },
    ...
}
```

actual

expected

```
| Title         | Trainer       | Duration  | Skill Level
|
| ------------- | ------------- | --------- | -------------
|
| Go            | Ben           | 4 hours   | Beginner
|
| Java Threads  | John          | 8 hours   | Advanced
|
| XML           | Mary          | 2 hours   | Beginner
```

What’s a Golden File?

Externalized expectation into a file

```
$ tree testdata
testdata
└── table
    └── trainings.golden
```

trainings.golden

```
| Title         | Trainer       | Duration   | Skill Level
|
| ------------- | ------------- | ---------- | -------------
|
| Go            | Ben           | 4 hours    | Beginner
|
| Java Threads  | John          | 8 hours    | Advanced
|
| XML           | Mary          | 2 hours    | Beginner
```

Updating Golden Files

You can build in functionality to update files on demand

```
$ go test ./... -update
```

Update expected values
based on actual output

Caution!

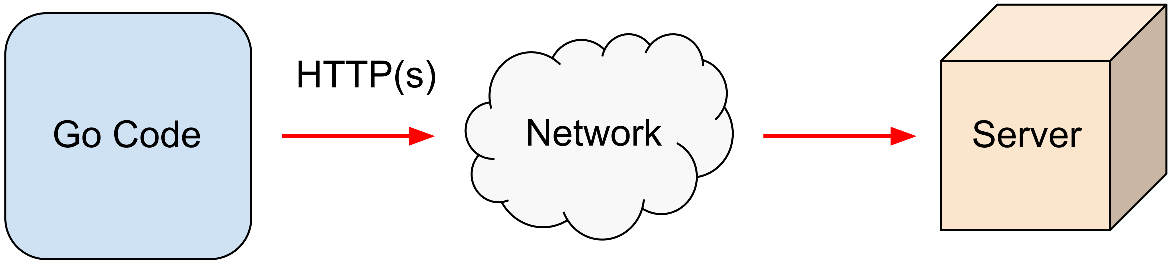
```
| Title         | Trainer       | Duration  | Skill Level   | Time
|
| ------------- | ------------- | --------- | ------------- | -------
|
| Go            | Ben           | 4 hours   | Beginner      | 8:00am
|
| Java Threads  | John          | 8 hours   | Advanced      | 7:30pm
|
```

EXERCISE

Using a Golden File
for JSON Data

Testing HTTP Services

“How can I test code without having the service running?”



Production Source Code

Typical usage of the HTTP client

```go
type HTTPGetter struct {
    client *http.Client
}

func (g *HTTPGetter) Get(href string) (*bytes.Buffer, error) {
    return g.get(href)
}

func (g *HTTPGetter) get(href string) (*bytes.Buffer, error) {
    buf := bytes.NewBuffer(nil)
    req, err := http.NewRequest("GET", href, nil)
    if err != nil {
        return buf, err
    }
    resp, err := g.client.Do(req)
    ...
}
```

Standard HTTP client implementation

Perform HTTP call

Emulating HTTP response in Test

The package `net/http` *provides helpful functionality*

```go
import (
    "net/http"
    "net/http/httptest"
)

func TestHTTPGet(t *testing.T) {
    server := httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.WriteHeader(200)
        w.Write([]byte("Expected Response"))
    }))
    defer server.Close()

    g := &HTTPGetter{client: &http.Client{}}
    data, err := g.Get(server.URL)
    ...
}
```

Test Server

Expected Response

EXERCISE

Avoiding Network Calls and Emulating the Response

Testable CLI-Layer Logic

“Do I need to run my program to test it?”

```
$ go run version
MyApp 1.2.3
```

Command

```
$ go run template download
Downloading template...
```

Subcommand

Using Cobra CLI Library

Widely-used option that enabled testing

https://github.com/spf13/cobra

- **Args:** What are we working on?
- **Command:** Action to execute
- **Flag:** Modifier for action

Cobra Command Definition

Specifies command options and description

```go
type versionCmd struct {
    out io.Writer
}

func newVersionCmd(out io.Writer) *cobra.Command {
    version := &versionCmd{out: out,}

    cmd := &cobra.Command{
        Use:   "version",
        Short: "print the version number and exit",
        RunE: func(cmd *cobra.Command, args []string) error {
            ...
            return version.run()
        },
    }
    return cmd
}
```

Inputs needed in command

Command definition

Executes command logic

Testable Command Logic

Command implementation that renders application version

```go
var version string

func SetVersion(v string) {
    version = v
}

func (v *versionCmd) run() error {
    _, err := fmt.Fprintf(v.out,"MyApp %s\n", version)
    if err != nil {
        return err
    }
    return nil
}
```

Version can be provided via linker option `-ldflags`

Pass version message to Writer

Test Implementation

No need to execute program, just run the relevant logic

```
func TestSemanticVersion(t *testingT) {
    b := bytes.NewBuffer(nil)
    SetVersion("1.2.3")
    version := &versionCmd{
      out: b,
    }
    err := version.run()

    assert.Nil(t, err)
    assert.Equal(t, "MyApp 1.2.3\n", b.String())
}
```

Capture output

Execute command logic

EXERCISE

Testing a Cobra CLI Command

Q & A

5 mins

Summary

Let’s wrap up what we’ve learnt...

O'REILLY®

Thank you