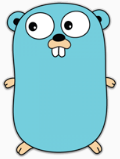
Programming in Go

Matt Holiday
Christmas 2020



Code Coverage

Testing culture

You should assume your code doesn’t work unless

- you have tests (unit, integration, etc.)
- they work correctly
- you run them
- they pass

Your change isn’t done until you’ve added / updated the tests

The correct order is compile, format, lint, test and *then* commit

This is basic code hygiene: **start clean, stay clean**

Testing JSON

```go
func contains(known, unknown map[string]interface{}) error {
    for k, v := range known {
        switch x := v.(type) {
        case float64:
            if !matchNum(k, x, unknown) {
                return fmt.Errorf("%s unmatched (%d)",
                                  k, int(x))
            }

        case string:
            if !matchString(k, x, unknown) {
                return fmt.Errorf("%s unmatched (%s)", k, x)
            }

        . . .
```

Testing JSON

```go
        . . .

        case map[string]interface{}:
            if val, ok := unknown[k]; !ok {
                return fmt.Errorf("%s missing in resp", k)
            } else if unk, ok := val.(map[string]interface{}); ok {
                if err := contains(x, unk); err != nil {
                    return fmt.Errorf("%s unmatched (%+v): %s",
                                          k, x, err)
                }
            } else {
                return fmt.Errorf("%s wrong in resp (%#v)",
                                      k, val)
            }
        }
    }

    return nil
}
```

Testing JSON

```go
func CheckData(known string, unknown []byte) error {
    var k, u map[string]interface{}

    if err := json.Unmarshal([]byte(known), &k); err != nil {
        return err
    }

    if err := json.Unmarshal(unknown, &u); err != nil {
        return err
    }

    return contains(k, u)
}
```

Testing JSON

Run the tests and analyze the code coverage

```go
// go test -v
// go test ./... -cover
// go test ./... -coverprofile=c.out -covermode=count
// go tool cover -html=c.out

var unknown = `{
    "id": 1,
    "name": "bob",
    "addr": {
        "street": "Lazy Lane",
        "city": "Exit",
        "zip": "99999"
    },
    "extra": 21.1
}`
```

Testing JSON

```go
func TestContains(t *testing.T) {
    var known = []string{
        `{"id": 1}`,
        `{"extra": 21.1}`,
        `{"name": "bob"}`,
        `{"addr": {"street": "Lazy Lane", "city": "Exit"}}`,
    }

    for _, k := range known {
        if err := CheckData(k, []byte(unknown)); err != nil {
            t.Errorf("invalid: %s (%s)\n", k, err)
        }
    }
}
```

Testing JSON

```
func TestNotContains(t *testing.T) {
    var known = []string{
        `{"id": 2}`,
        `{"pid": 2}`,
        `{"name": "bobby"}`,
        `{"first": "bob"}`,
        `{"addr": {"street": "Lazy Lane", "city": "Alpha"}}`,
    }

    for _, k := range known {
        if err := CheckData(k, []byte(unknown)); err == nil {
            t.Errorf("false positive: %s\n", k)
        } else {
            t.Log(err)
        }
    }
}
```

Code coverage

Running `go test -cover` finds what part of the code is exercised by the unit tests

```
$ go test -cover
PASS
coverage: 85.2% of statements
```

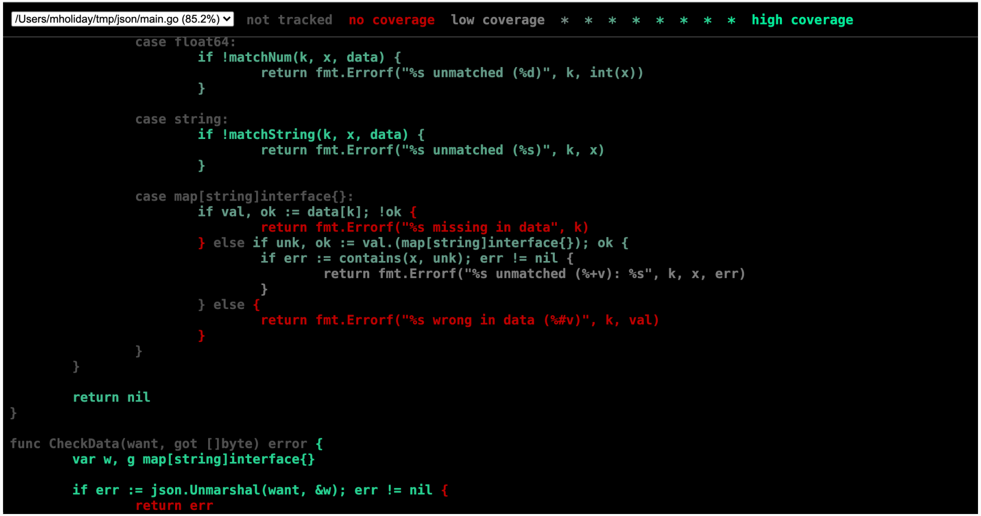
Using the `-coverprofile` flag generates a file with coverage counts

This can be passed to another tool to display coverage visually

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

Using the `-covermode=count` flag turns it into a heat map

Code coverage


/Users/mholiday/tmp/json/main.go (85.2%)
not tracked no coverage low coverage * * * * * * * * high coverage

```
                case float64:
                        if !matchNum(k, x, data) {
                                return fmt.Errorf("%s unmatched (%d)", k, int(x))
                        }

                case string:
                        if !matchString(k, x, data) {
                                return fmt.Errorf("%s unmatched (%s)", k, x)
                        }

                case map[string]interface{}:
                        if val, ok := data[k]; !ok {
                                return fmt.Errorf("%s missing in data", k)
                        } else if unk, ok := val.(map[string]interface{}); ok {
                                if err := contains(x, unk); err != nil {
                                        return fmt.Errorf("%s unmatched (%+v): %s", k, x, err)
                                }
                        } else {
                                return fmt.Errorf("%s wrong in data (%#v)", k, val)
                        }
                }
        }

        return nil
}

func CheckData(want, got []byte) error {
        var w, g map[string]interface{}

        if err := json.Unmarshal(want, &w); err != nil {
                return err
```

Code coverage

The heat map shows two cases we haven't covered:

- The case where the key is missing
- The case where it has the wrong type (not an object)

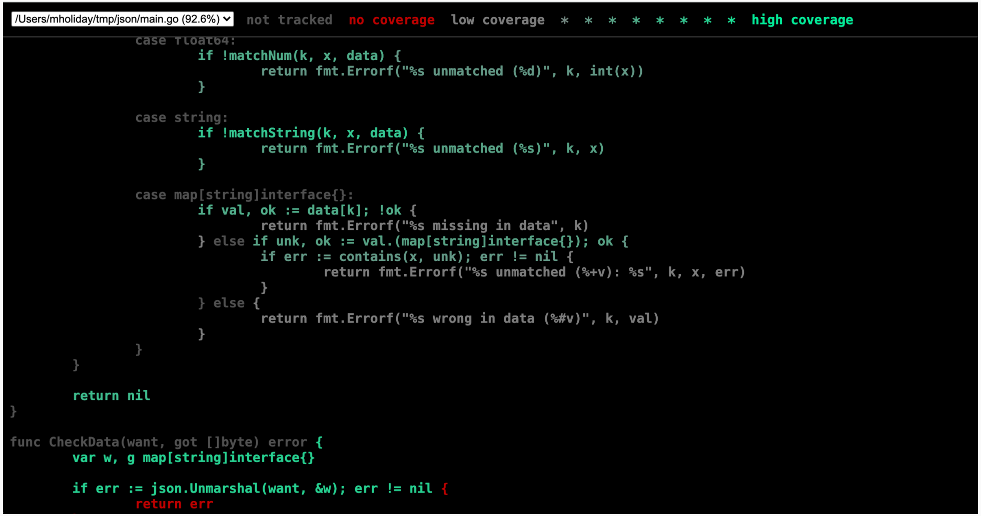
We need to add some more subtests to cover this code

```
$ go test ./... -coverprofile=c.out -covermode=count
ok   _/Users/mholiday/tmp/json 0.173s coverage: 92.6% of statements

$ go tool cover -html=c.out
```

Testing JSON

```
func TestNotContains(t *testing.T) {
    var known = []string{
        `{"id": 2}`,
        `{"pid": 2}`,
        `{"name": "bobby"}`,
        `{"first": "bob"}`,
        `{"addr": {"street": "Lazy Lane", "city": "Alpha"}}`,
        `{"city": {"avenue": "Lazy Ave"}}`,                  // missing
        `{"name": {"avenue": "Lazy Ave"}}`,                  // wrong
    }

    . . .
}
```

Code coverage


/Users/mholiday/tmp/json/main.go (92.6%) not tracked no coverage low coverage * * * * * * * * high coverage

```
                case float64:
                        if !matchNum(k, x, data) {
                                return fmt.Errorf("%s unmatched (%d)", k, int(x))
                        }

                case string:
                        if !matchString(k, x, data) {
                                return fmt.Errorf("%s unmatched (%s)", k, x)
                        }

                case map[string]interface{}:
                        if val, ok := data[k]; !ok {
                                return fmt.Errorf("%s missing in data", k)
                        } else if unk, ok := val.(map[string]interface{}); ok {
                                if err := contains(x, unk); err != nil {
                                        return fmt.Errorf("%s unmatched (%+v): %s", k, x, err)
                                }
                        } else {
                                return fmt.Errorf("%s wrong in data (%#v)", k, val)
                        }
                }
        }

        return nil
}

func CheckData(want, got []byte) error {
        var w, g map[string]interface{}

        if err := json.Unmarshal(want, &w); err != nil {
                return err
```