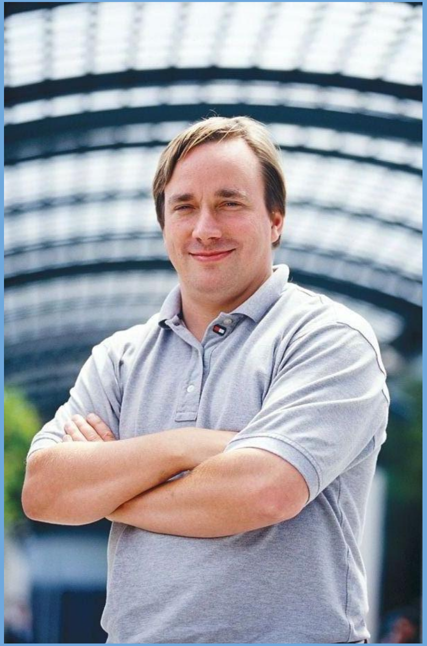
Unit тестирование в Go.

Tips and Tricks

+ **Workshop**

Сергей Иваненко

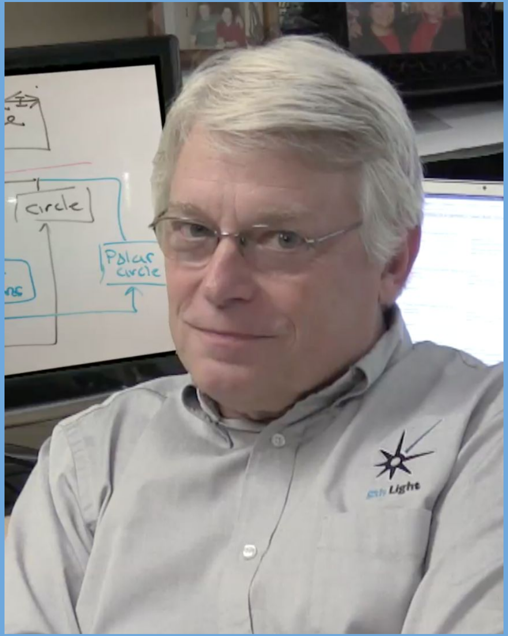


Тестирование?!
Если оно компилируется - хорошо, если загружается - великолепно!

Linus Torvalds

Я плачу за код, который работает, а не за тестирование.

Kent Beck



Профессиональные разработчики тестируют свой код.

Robert Martin

Будь как ORACLE®

Создание стартапа в Кремниевой долине

Инженеры Ларри Эллисон (Larry Ellison), Боб Майнер (Bob Miner) и Эд Оутс (Ed Oates) основали компанию Software Development Laboratories. Их первый офис имел площадь менее 100 кв. метров и находился в городе Санта-Клара, Калифорния.

1977

Этой базе данных **43 года,**
а она до сих пор в деле!

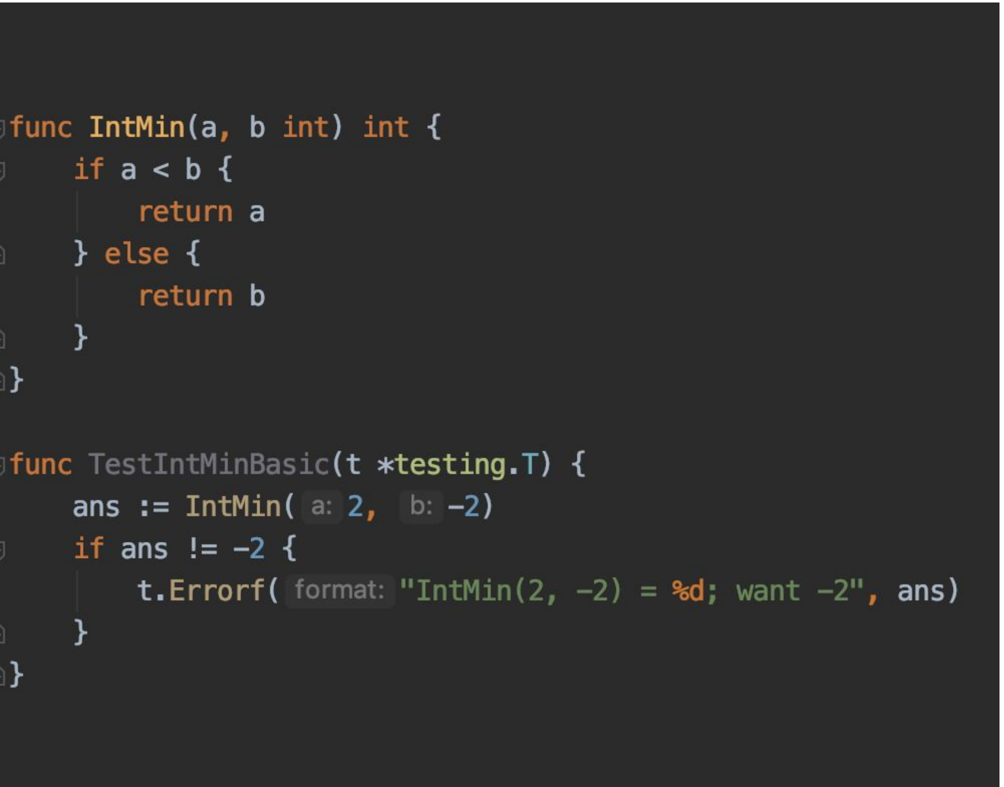
golang.org/pkg/testing/

- func TestXxx(*testing.T)
- func BenchmarkXXX(*testing.B)
- Subtests and Sub-benchmarks
- Table tests

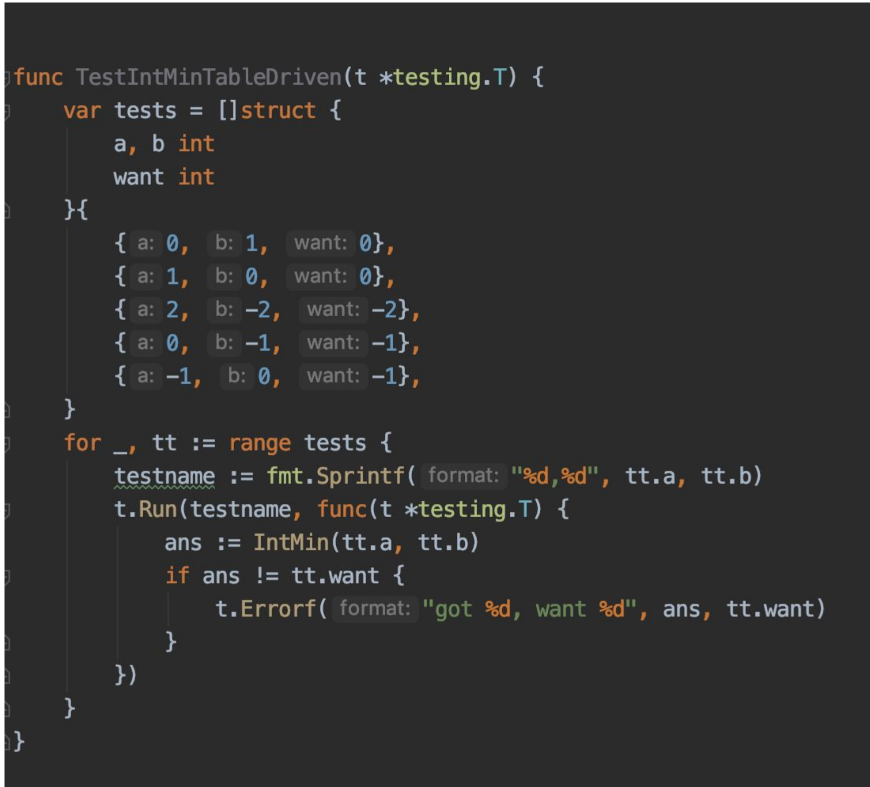
Пакеты и тесты

- Функция и тест в одном пакете
- Функция и тест в независимых пакетах

Google пример



```go
func IntMin(a, b int) int {
    if a < b {
        return a
    } else {
        return b
    }
}

func TestIntMinBasic(t *testing.T) {
    ans := IntMin( a: 2,  b: -2)
    if ans != -2 {
        t.Errorf( format: "IntMin(2, -2) = %d; want -2", ans)
    }
}
```



```go
func TestIntMinTableDriven(t *testing.T) {
    var tests = []struct {
        a, b int
        want int
    }{
        { a: 0,  b: 1,  want: 0},
        { a: 1,  b: 0,  want: 0},
        { a: 2,  b: -2,  want: -2},
        { a: 0,  b: -1,  want: -1},
        { a: -1,  b: 0,  want: -1},
    }
    for _, tt := range tests {
        testname := fmt.Sprintf( format: "%d,%d", tt.a, tt.b)
        t.Run(testname, func(t *testing.T) {
            ans := IntMin(tt.a, tt.b)
            if ans != tt.want {
                t.Errorf( format: "got %d, want %d", ans, tt.want)
            }
        })
    }
}
```

Важные замечания

- DI
- accepts interface returns structures

Файловый пример

- Тип
- Контент
- Права
- и т.п

github.com/spf13/afero

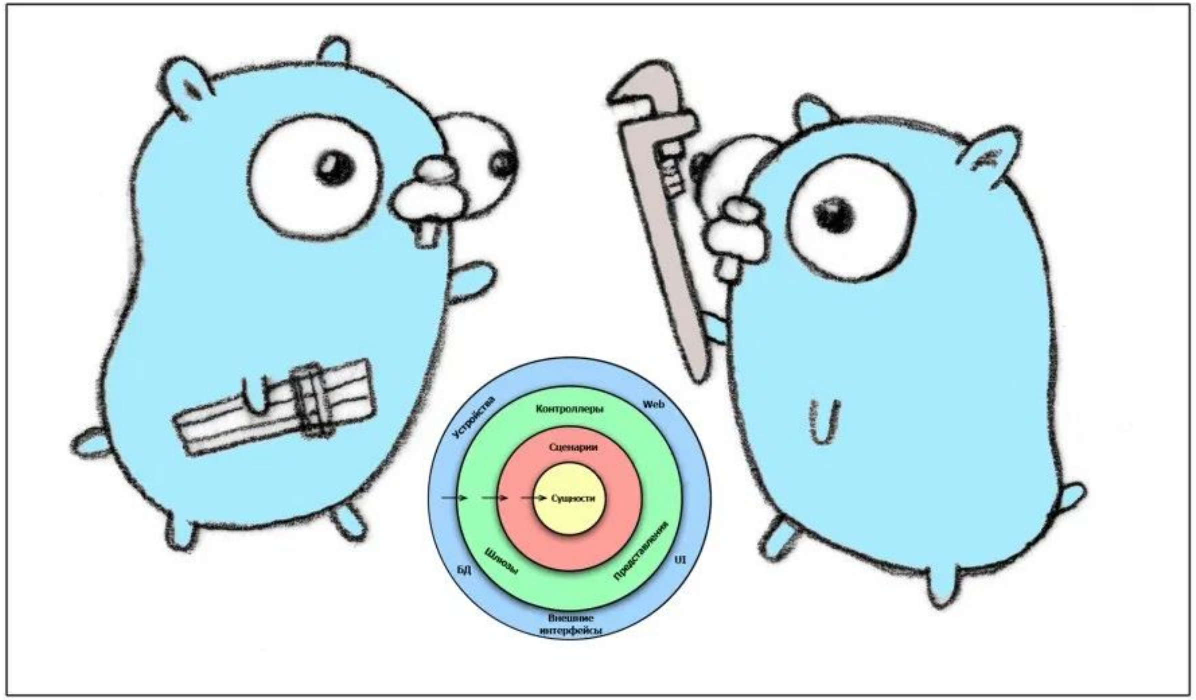
```go
func Create(fs afero.Fs, filename string) error {
  handler, err := fs.Create(filename)
  if err != nil {
     return err
  }
  defer handler.Close()
  return nil
}
```

```go
type Fs interface {
  Create(name string) (File, error)
  Mkdir(name string, perm os.FileMode) error
  MkdirAll(path string, perm os.FileMode) error
```

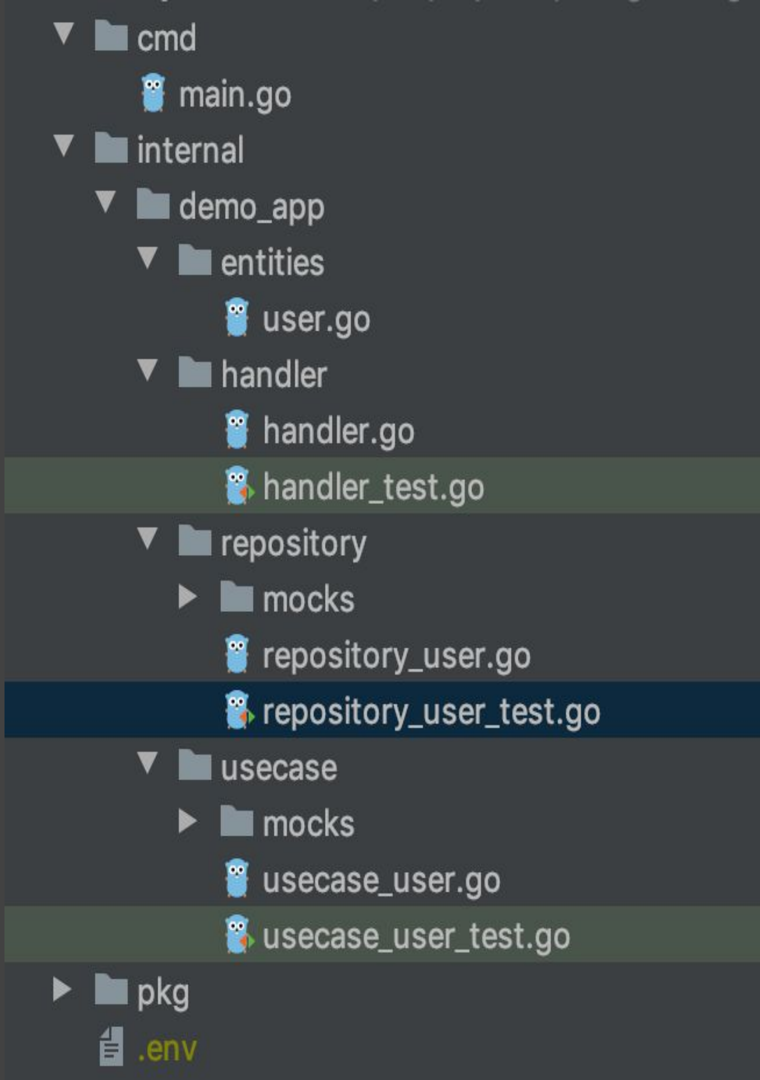
```go
func Test_Create(t *testing.T) {
  type args struct {
    fs       afero.Fs
    filename string
  }
  tests := []struct {
    name    string
    args    args
    wantErr bool
  }{
    {
      name: "success",
      args: args{
        fs:       afero.NewMemMapFs(),
        filename: "somefile.txt",
      },
      wantErr: false,
    },
    {
      name: "false",
      args: args{
        fs:       afero.NewOsFs(),
        filename: "forbiden/wriring/somefile.txt",
      }
      wantErr: true,
    },
   }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      if err := files.Create(tt.args.fs, tt.args.filename); (err != nil) != tt.wantErr {
        t.Errorf("Create() error = %v, wantErr %v", err, tt.wantErr)
      }
    })
  }
}
```

Http + use case + db + unit tests

- mockery
- sqlmock
- testify
- generate



```go
func main() {
  router := mux.NewRouter()

  db := postgres.Connection
  repo := repository.New(context.Background(), db)
  useCase := usecase_user.New(repo)
  h := handler.New(useCase)

  router.HandleFunc("/user", h.GetUser).Methods(http.MethodGet)
  router.HandleFunc("/user/add", h.AddUser).Methods(http.MethodPost)
  router.HandleFunc("/user/edit", h.EditUser).Methods(http.MethodPost)
  router.HandleFunc("/user/delete", h.RemoveUser).Methods(http.MethodPost)

  if err := http.ListenAndServe(":8080", router); err != nil {
     panic(err.Error())
  }
}
```



```
▼ cmd
    main.go
▼ internal
  ▼ demo_app
    ▼ entities
        user.go
    ▼ handler
        handler.go
        handler_test.go
    ▼ repository
      ▶ mocks
        repository_user.go
        repository_user_test.go
    ▼ usecase
      ▶ mocks
        usecase_user.go
        usecase_user_test.go
▶ pkg
  .env
```

```go
func (h *handler) AddUser(w http.ResponseWriter, r *http.Request) {
  w.Header().Set(`Content-type`, `application/json`)
 type payload struct {
    Name string `json:"name"`
    Email string `json:"email"`
  }

  p := new(payload)
  if err := json.NewDecoder(r.Body).Decode(&p); err != nil {
    http.Error(w,
                http.StatusText(http.StatusBadRequest),
                http.StatusBadRequest)

    return
  }

  user, err := h.us.Create(p.Name, p.Email)
  if err != nil {
    http.Error(w,
                http.StatusText(http.StatusInternalServerError),
                http.StatusInternalServerError)

    return
  }

  json.NewEncoder(w).Encode(user)
}
```

```go
type args struct {
  w *httptest.ResponseRecorder
  r  *http.Request
}

func setUpArgs(method string, route string, payload string) args {
  req, err := http.NewRequest(
                                        method,
                                        route,
                                        strings.NewReader(payload)
                    )
  if err != nil {
    log.Fatal(err.Error())
  }

  return args{
    w: httptest.NewRecorder(),
    r: req,
  }
}
```

```go
func Test_handler_AddUser(t *testing.T) {
  route := `/new/user`

  type fields struct {
     us usecase_user.User
  }
  tests := []struct {
     name              string
     fields            func() fields
     args              args
     wantStatusCode int
     wantBody          string
  } {
      {},
      //...другие сценарии
  }


   for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
     h := http.HandlerFunc(handler.New(tt.fields().us).AddUser)
     h.ServeHTTP(tt.args.w, tt.args.r)

     assert.Equal(t, tt.wantStatusCode, tt.args.w.Code)
     assert.Equal(t, tt.wantBody, tt.args.w.Body.String())
  })
}
```

```go
{
  name: `Success`,
  fields: func() fields {

     mock := mocks.User{} // Это mock user usecase
     mock.On(`Create`, `gopher`, `gopher@kalinigrad.ru`).Return(
        entities.User{
                          ID:    1,
                          Name:  "gopher",
                          Email:   "gopher@kalinigrad.ru"},
      nil)

     return fields{
        us: &mock,
     }
  },
  args:  setUpArgs(
                 http.MethodPost,
                 route,
                 `{"name": "gopher", "email": "gopher@kalinigrad.ru"}`),



  wantStatusCode: http.StatusOK,
  wantBody:          `{"id":1,"name":"gopher","email":"gopher@kalinigrad.ru"}`,
},
//go:generate mockery -name=User
type (
  User interface {
     Get(ctx context.Context, id int64) (entities.User, error)
     Create(ctx context.Context, name, email string) (entities.User,
error)
     UpdateEmail(ctx context.Context, id int64, email string)
(entities.User, error)
     Delete(ctx context.Context, id int64) error
  }
  usecase struct {
     repo repository.User
  }
)

func (u *usecase) Get(ctx context.Context, id int64) (entities.User,
error) {
  return u.repo.Read(id)
}

func (u *usecase) Create(ctx context.Context,name, email string)
(entities.User, error) {
  return u.repo.Create(name, email)
}

func (u *usecase) UpdateEmail(ctx context.Context, id int64, email
string) (entities.User, error) {
  return u.repo.UpdateEmail(id, email)
}

func (u *usecase) Delete(id int64) error {
  return u.repo.Delete(id)
```

```go
// User is an autogenerated mock type for the User type
type User struct {
  mock.Mock
}

// Create provides a mock function with given fields: ctx, name, email
func (_m *User) Create(ctx context.Context, name string, email string)
(entities.User, error) {
  ret := _m.Called(ctx, name, email)

  var r0 entities.User
  if rf, ok := ret.Get(0).(func(context.Context, string, string) entities.User); ok {
     r0 = rf(ctx, name, email)
  } else {
     r0 = ret.Get(0).(entities.User)
  }

  var r1 error
  if rf, ok := ret.Get(1).(func(context.Context, string, string) error); ok {
     r1 = rf(ctx, name, email)
  } else {
     r1 = ret.Error(1)
  }

  return r0, r1
}
```

```go
func Test_usecase_Create(t *testing.T) {
  type fields struct {
    repo repository.User
  }
  type args struct {
    ctx context.Context,
    name  string
    email   string
  }
  tests := []struct {
    name    string
    fiedls     fields //Обычная генерилка
    fields     func(args args, want entities.User) fields
    args       args
    want      entities.User
    wantErr bool
  }

for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
    repo := tt.fields(tt.args, tt.want).repo
    u := usecase_user.New(repo)

    got, err := u.Create(tt.args.name, tt.args.email)
    if (err != nil) != tt.wantErr {
      t.Errorf("Create() error = %v, wantErr %v", err, tt.wantErr)
      return
    }
    if !reflect.DeepEqual(got, tt.want) {
      t.Errorf("Create() got = %v, want %v", got, tt.want)
    }
```

```go
{
  name: `Success`,
  fields: func(args args, want entities.User) fields {
    m := &mocks.User{} // repo_mock
    m.On(`Create`, args.ctx, args.name, args.email).Return(want, nil)

    return fields{
      repo: m,
    }
  },
  args: args{
    name:  "gopher",
    email:  "gopner@kalinigrad.ru",
  },
  want: entities.User{
    ID:    1,
    Name:  `gopher`,
    Email:  `gopher@kalinigrad.ru`,
  },
  wantErr: false,
},
//go:generate mockery -name=User
type (
  User interface {
    Create(ctx context.Context, name, email string) (entities.User, error)
    Read(ctx context.Context, id int64) (entities.User, error)
    UpdateEmail(ctx context.Context, id int64, email string) (entities.User, error)
    Delete(ctx context.Context, id int64) error
  }

  repo struct {
    db *sqlx.DB
  }
)

func (r *repo) Create(ctx context.Context, name, email string) (entities.User, error) {
  var user entities.User
  err := r.db.GetContext(ctx, &user,
   `INSERT INTO demo.public.users (name, email) VALUES  ($1, $2)   RETURNING *`, name, email)

  return user, err
}
```

```go
func Test_repo_Create(t *testing.T) {
  queryTml := `INSERT INTO demo.public.users (name, email) VALUES  ($1,
$2) RETURNING *`
  rowsTml := []string{`id`, `name`, `email`}

  type fields struct {
     sqlx  *sqlx.DB
  }
  type args struct {
    ctx     contex.Context
    name  string
    email  string
  }
  tests := []struct {
    name     string
    fiedls     fields //Обычная генерилка
    fields  func(args args, db *sql.DB, mock sqlmock.Sqlmock,
                     want entities.User) fields
    args      args
    want      entities.User
    wantErr  bool
  }
for _, tt := range tests {
 // левая часть слайда
}
```

```go
t.Run(tt.name, func(t *testing.T) {
    db, sqmock, _ :=
sqlmock.New(sqlmock.QueryMatcherOption(sqlmock.QueryMatcherEqu
al))

    fields := tt.fields(tt.args, db, sqmock, tt.want)
    r := repository.New(fields.sqlx)

    got, err := r.Create(tt.args.ctx, tt.args.name, tt.args.email)

   if (err != nil) != tt.wantErr {
      t.Errorf("Create() error = %v, wantErr %v", err, tt.wantErr)
      return
    }
    if !reflect.DeepEqual(got, tt.want) {
      t.Errorf("Create() got = %v, want %v", got, tt.want)
    }
  })
```

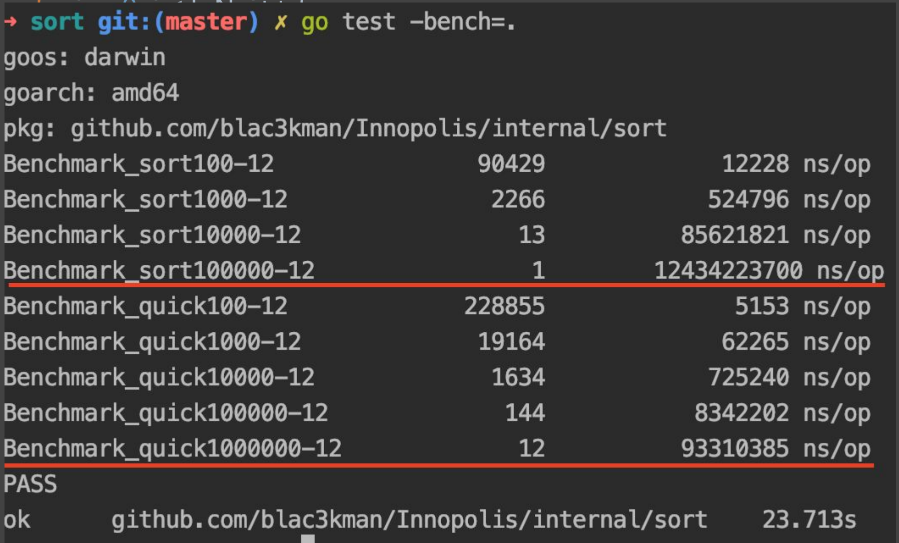
```go
{
  name: "Success",
  args: args{
    ctx:     context.TODO()
    name:  "gopher",
    email:   "gopher@kalinigrad.ru",
  },
  want: entities.User{
    ID:       1,
    Name:  "gopher",
    Email:   "gopher@kalinigrad.ru",
  },
  wantErr: false,
  fields: func(args args, db *sql.DB, mock sqlmock.Sqlmock,
                  want entities.User) fields {
    rows := sqlmock.NewRows(rowsTml)

    rows.AddRow(
      want.ID,
      want.Name,
      want.Email,
    )

    mock.ExpectQuery(queryTml).WithArgs(args.name, args.email).
      WillReturnRows(rows)

   return fields{  sqlx: sqlx.NewDb(db, "sqlmock") }
  },
},
```

```go
db, sqmock, err :=
sqlmock.New(sqlmock.QueryMatcherOption(sqlmock.QueryMatcherEqual))
if err != nil {
}

fields := tt.fields(tt.args, db, sqmock, tt.want)
r := repository.New(fields.sqlx)

got, err := r.Create(tt.args.ctx, tt.args.name, tt.args.email)
```

go test -bench=.

```
func Benchmark_sort100(b *testing.B) {
  for i := 0; i < b.N; i++ {
     bubble(generateSlice(hundred))
  }
}

func Benchmark_sort1000(b *testing.B) {
  for i := 0; i < b.N; i++ {
     bubble(generateSlice(oneThousand))
  }
}

func Benchmark_sort10000(b *testing.B) {
  for i := 0; i < b.N; i++ {
     bubble(generateSlice(tenThousands))
  }
}

func Benchmark_sort100000(b *testing.B) {
  for i := 0; i < b.N; i++ {
     bubble(generateSlice(hundredThousands))
  }
}
```

```
func Benchmark_quick100(b *testing.B) {
  for i := 0; i < b.N; i++ {
     quick(generateSlice(hundred))
  }
}

func Benchmark_quick1000(b *testing.B) {
  for i := 0; i < b.N; i++ {
     quick(generateSlice(oneThousand))
  }
}

func Benchmark_quick10000(b *testing.B) {
```



```
→ sort git:(master) ✗ go test -bench=.
goos: darwin
goarch: amd64
pkg: github.com/blac3kman/Innopolis/internal/sort
Benchmark_sort100-12                90429           12228 ns/op
Benchmark_sort1000-12                2266          524796 ns/op
Benchmark_sort10000-12                 13        85621821 ns/op
Benchmark_sort100000-12                 1     12434223700 ns/op
Benchmark_quick100-12              228855            5153 ns/op
Benchmark_quick1000-12              19164           62265 ns/op
Benchmark_quick10000-12              1634          725240 ns/op
Benchmark_quick100000-12              144         8342202 ns/op
Benchmark_quick1000000-12              12        93310385 ns/op
PASS
ok      github.com/blac3kman/Innopolis/internal/sort    23.713s
```

```
go test -bench=. sort_cmp_test.go > quick.txt
go test -bench=. sort_cmp_test.go > bubble.txt
```

```go
func Sort(a []int64) {
  bubble(a)
}

func Sort(a []int64) {
  quick(a)
}
```

```
benchcmp quick.txt bubble.txt
```

```
benchmark                  old ns/op    new ns/op      delta
Benchmark_Sort100000-12    8547238      12957265671    +151495.94%
```

Тестирование полезно и увлекательно

Спасибо!

blac3kman/Innopolis

@SergeyWh1te