

# Go 테스트의 거의 모든 것

실습으로 쉽게 알아보자Go

김정민 / 삼성SDS



# 발표자

김정민

관심언어: Go, JavaScript, PHP, Python, Java, C++

Samsung Kubernetes Engine 개발

오픈소스 활동

- Lethe (Log DB)
- Venti (Visualizer)

<https://github.com/kuoss>

참여(문의/PR/리뷰) 환영합니다

# 들어가며...

**발표자료**    <https://github.com/jimnote/go-test> (문서 & 코드)

- 실습용 코드 (핸즈온은 아님)
- go 1.20.5에서 작동 확인

**참고자료**

- Go 블로그
- go help test
- testing, testify 문서 등

※ 맨 뒤 ‘참고자료’ 참고

# 목차

## I. Go 테스트 기초

1. Go 테스트 시작하기
2. 라이브러리 & testify
3. Whitebox & Blackbox Test
4. 서브테스트 & 테이블 드리븐

## II. Go 테스트 기능 활용

5. 테스트의 유형 & Benchmark
6. Fuzz & Example
7. Failfast & Skip
8. Parallel & race

## III. 테스트 사례

9. 변화하는 것에 대한 테스트
10. Error & Panic 테스트
11. HTTP 테스트

## IV. mock & 수명주기

12. mock
13. HTTP서버 mock
14. TestMain & suite

## V. IDE & 커버리지

15. vscode 활용
16. 커버리지
17. 커버리지 100%?

## VI. 기타

18. 통합 테스트
19. 테스트 이외의 테스트?
20. 참고자료



# I. Go 테스트 기초

- 1. Go 테스트 시작하기
- 2. 라이브러리 & testify
- 3. Whitebox & Blackbox Test
- 4. 서브테스트 & 테이블 드리븐



# 1. Go 테스트 시작하기



# 시작하기 #1

파일명: xxx\_test.go

패키지명: 주로 xxx (가끔 xxx\_test)

- xxx      화이트박스 테스트 (같은 패키지명)
- xxx\_test   블랙박스 테스트

함수명: TestXxx (Xxx는 주로 테스트 대상 함수명)

- Testxxx    사용 불가. 테스트 함수로 인식되지 않음
- Test\_xxx   권장하지 않지만, 사용은 가능

아무 것도 테스트하지 않지만  
오류 없이 통과하는...  
간단한 테스트 코드

```
package xxx
```

```
import "testing"
```

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

실패(FAIL) 코드 없으므로, 통과(PASS).  
테스트 코드가 원래 그런 식이다.

## 시작하기 #2   테스트하는 것이 있는 간단한 예시

```
// myabs.go  
package myabs
```

```
func Abs(num int) int {  
    if num < 0 {  
        return -num  
    }  
    return num  
}
```

```
// myabs_test.go  
package myabs
```

```
import (  
    "testing"  
)
```

```
func TestAbs(t *testing.T) {  
    got := Abs(-2)  
    if got != 2 {  
        t.Errorf("Abs(-2) = %d; want 2", got)  
    }  
}
```



# 시작하기 #3 go test 실행 (PASS 예시)

PROBLEMS OUTPUT TEST RESULTS TERMINAL DEBUG CONSOLE

PASS인 경우... exit status 0

```
root@aws1:~/go/src/go-test/myabs# go test
PASS
ok      github.com/jmnote/go-test/myabs 0.001s
root@aws1:~/go/src/go-test/myabs# go test .
ok      github.com/jmnote/go-test/myabs (cached) 캐시됨
root@aws1:~/go/src/go-test/myabs# go test ./... 하위 전체
ok      github.com/jmnote/go-test/myabs (cached)
root@aws1:~/go/src/go-test/myabs# go test
PASS
ok      github.com/jmnote/go-test/myabs 0.001s
root@aws1:~/go/src/go-test/myabs#
```

verbose

```
root@aws1:~/go/src/go-test/myabs# go test -v
=== RUN   TestAbs
--- PASS: TestAbs (0.00s) 테스트 함수명 (수행시간)
PASS
ok      github.com/jmnote/go-test/myabs 0.001s
root@aws1:~/go/src/go-test/myabs# go test . -v
=== RUN   TestAbs
--- PASS: TestAbs (0.00s)
PASS
ok      github.com/jmnote/go-test/myabs (cached)
root@aws1:~/go/src/go-test/myabs# go test -v ./...
=== RUN   TestAbs
--- PASS: TestAbs (0.00s)
PASS
ok      github.com/jmnote/go-test/myabs (cached)
root@aws1:~/go/src/go-test/myabs#
```

## 시작하기 #4 로컬 디렉토리 모드, 패키지 목록 모드

구분	로컬 디렉토리 모드 (패키지 미지정)	패키지 목록 모드 (패키지 지정)
명령어 예시	\$ go test	\$ go test . \$ go test ./... \$ go test github.com/xxx/xxx
캐시	X	O (단, PASS인 경우)
기타	-	IDE에서는 이 모드 사용

- 대부분 IDE에서 테스트하기 때문에, 이 구분이 아주 중요하지는 않다.
- 가끔 CLI에서 실행할 때도 있으니, 이런 것이 있다는 정도는 알아두자.

# 시작하기 #5 go test 실행 (FAIL 예시)

```
// myabs2_test.go
package myabs2

import (
    "testing"
)

func TestAbs(t *testing.T) {
    got := Abs(-2)
    if got != 1 { FAIL하도록 변경
        t.Errorf("Abs(-2) = %d; want
2", got)
    }
}
```

```
❌ root@aws1:~/go/src/go-test/myabs2# go test
--- FAIL: TestAbs (0.00s)
    myabs2_test.go:10: Abs(-2) = 2; want 2
FAIL
exit status 1
FAIL    github.com/jmnote/go-test/myabs2    0.001s
❌ root@aws1:~/go/src/go-test/myabs2# go test ./... 패키지 목록
--- FAIL: TestAbs (0.00s) 모드이지만
    myabs2_test.go:10: Abs(-2) = 2; want 2 캐시 비활성화
FAIL
FAIL    github.com/jmnote/go-test/myabs2    0.001s
FAIL
❌ root@aws1:~/go/src/go-test/myabs2# go test ./... -v
=== RUN   TestAbs
    myabs2_test.go:10: Abs(-2) = 2; want 2
--- FAIL: TestAbs (0.00s)
FAIL
FAIL    github.com/jmnote/go-test/myabs2    0.001s
FAIL
● root@aws1:~/go/src/go-test/myabs2# echo $?
1 exit status 1
○ root@aws1:~/go/src/go-test/myabs2#
○ root@aws1:~/go/src/go-test/myabs2#
```



## 2. 라이브러리 & testify



# 테스트 라이브러리

Go 테스트 라이브러리는 다양하다...

<https://github.com/avelino/awesome-go#testing>

구분	testing ★	testify ★	ginkgo
인기	<b>표준</b> 라이브러리	서드파티 GitHub Star <b>20.1k</b>	서드파티 GitHub Star 7.3k
About	“Go 패키지 자동 테스트 지원”	“표준 라이브러리와 잘 작동하는 assertion, mock을 포함한 툴킷”	“Go 현대적 테스트 프레임워크”
특징	표준 라이브러리 if, DeepEqual 작성 필요	<b>간결함</b> (if, DeepEqual 작성 불필요), 유용한 하위패키지 ( <b>assert/require/mock/suite</b> )	BDD(behavior-driven development) 지원
사례	표준 라이브러리(에서 사용), testify	etcd, influxdb, gin, prometheus, kubernetes (단위 테스트)	kubernetes (e2e 테스트) <a href="https://github.com/kubernetes/kubernetes/blob/v1.27.4/hack/ginkgo-e2e.sh">https://github.com/kubernetes/kubernetes/blob/v1.27.4/hack/ginkgo-e2e.sh</a>
Repo	<a href="https://cs.opensource.google/go/go">https://cs.opensource.google/go/go</a>	<a href="https://github.com/stretchr/testify">https://github.com/stretchr/testify</a>	<a href="https://github.com/onsi/ginkgo">https://github.com/onsi/ginkgo</a>

# testing & testify

```
// myabs2_test.go
package myabs2

import (
    "testing"
)

func TestAbs(t *testing.T) {
    got := Abs(-2)
    if got != 1 {
        t.Errorf("Abs(-2) = %d; want 2", got)
    }
}
```

[testing]

if문/메시지 작성 필요

※ 복잡한 자료형에는 DeepEqual도 필요

```
// myabs3_test.go
package myabs3

import (
    "testing"
    "github.com/stretchr/testify/assert"
)

func TestAbs(t *testing.T) {
    got := Abs(-2)
    assert.Equal(t, 1, got)
}
```

[testify]

if/메시지/DeepEqual 불필요



# testify

```
func TestAbs(t *testing.T) {  
    got := Abs(-2)  
    assert.Equal(t, 1, got)  
}
```

```
root@wsl:~/go/src/go-test/myabs3# go test
```

```
--- FAIL: TestAbs (0.00s)
```

```
myabs3_test.go:11:
```

메시지 작성 없이도 상세 정보 표시

```
Error Trace: /root/go/src/go-test/myabs3/myabs3_test.go:12
```

```
Error: Not equal:
```

```
expected: 1
```

```
actual : 2
```

```
Test: TestAbs
```

※ 변수명 관례

- 기대한 값: **want**, expect(ed)

- 실제값: **got**, actual

```
FAIL
```

```
exit status 1
```

```
FAIL github.com/jmnote/go-test/myabs3
```

```
0.002s
```



## 3. Whitebox & Blackbox

패키지 내부에서 테스트할까? 외부에서 테스트할까?





# Whitebox & Blackbox #1

구분	화이트박스 테스트	블랙박스 테스트
설명	<ul style="list-style-type: none"><li>● 패키지명: xxx</li><li>● ‘일반적인 단위 테스트’</li><li>● 전체 코드에 대한 테스트</li><li>● exported &amp; unexported 테스트 가능</li><li>● 코드 커버리지에 관심</li></ul>	<ul style="list-style-type: none"><li>● 패키지명: xxx_test</li><li>● ‘라이브러리로서의 테스트’</li><li>● 외부에서 import하여 사용하는 테스트</li><li>● unexported는 테스트 불가</li><li>● 기능적 커버리지에 관심</li></ul>

블랙박스 테스트의 경우,

import하여 테스트하므로 다른 디렉토리(패키지)에 있어도 된다.

그래도 같은 곳에 있어야 찾기 쉬우므로, 특별한 이유가 없다면 같은 곳에 두자.

# Whitebox & Blackbox #2

## 테스트 대상

```
// mysquare.go
package mysquare

func Square(n int) int {
    return multiply(n, n)
} exported 함수

func multiply(a int, b int) int {
    return a * b
} unexported 함수
```

## 화이트박스 테스트

```
// mysquare_inner_test.go
package mysquare

import (
    ...
    ...
)

func TestSquare(t *testing.T) {
    got := Square(-2)
    assert.Equal(t, 4, got)
}

func TestMultiply(t *testing.T) {
    got := multiply(-2, -2)
    assert.Equal(t, 4, got)
}
```

## 블랙박스 테스트

```
// mysquare_outer_test.go
package mysquare_test

import (
    ... 패키지명 다르므로 import 필요
    "github.com/jmnote/go-test/mysquare"
)

func TestSquare(t *testing.T) {
    got := mysquare.Square(-2)
    assert.Equal(t, 4, got)
} exported는 테스트 가능

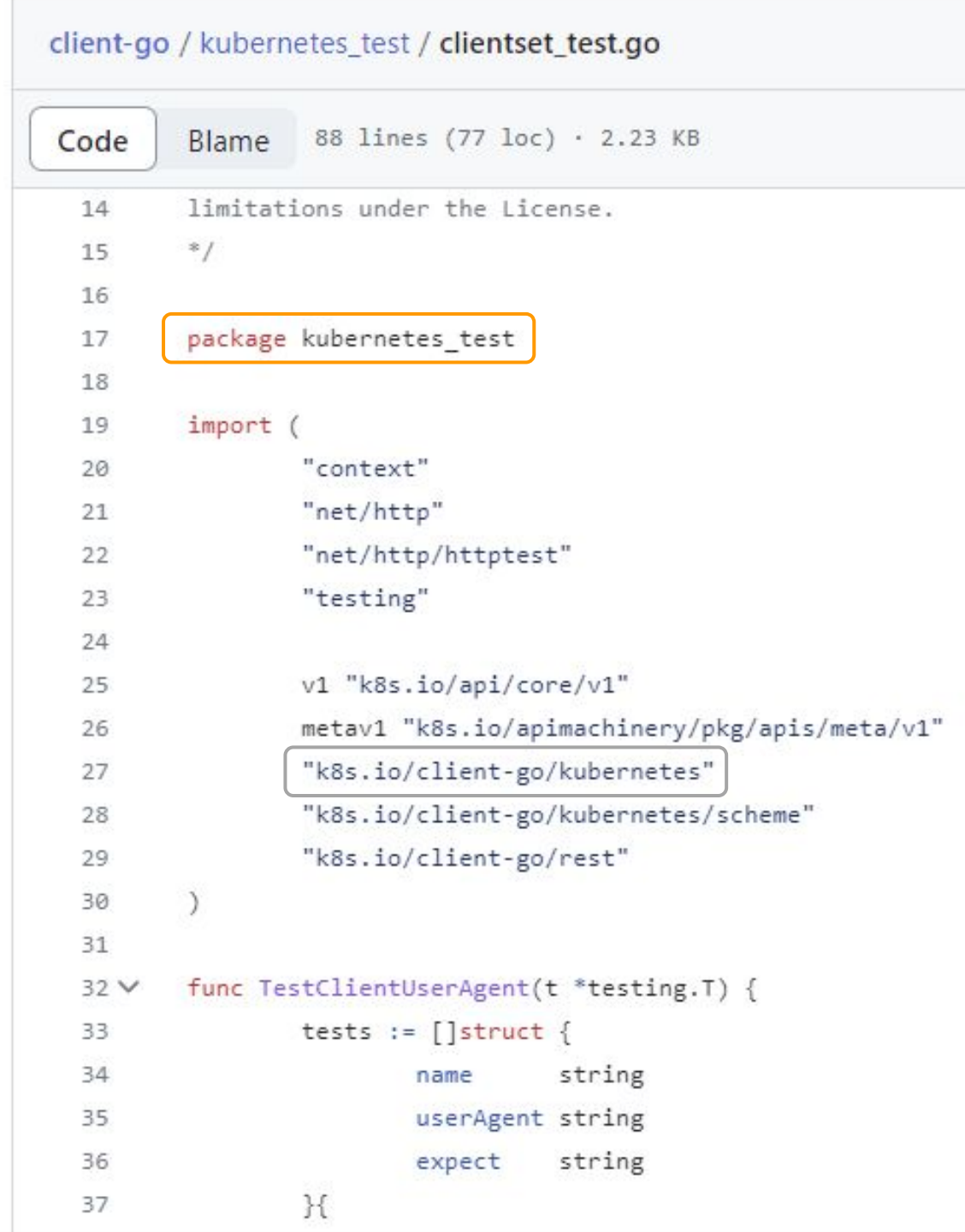
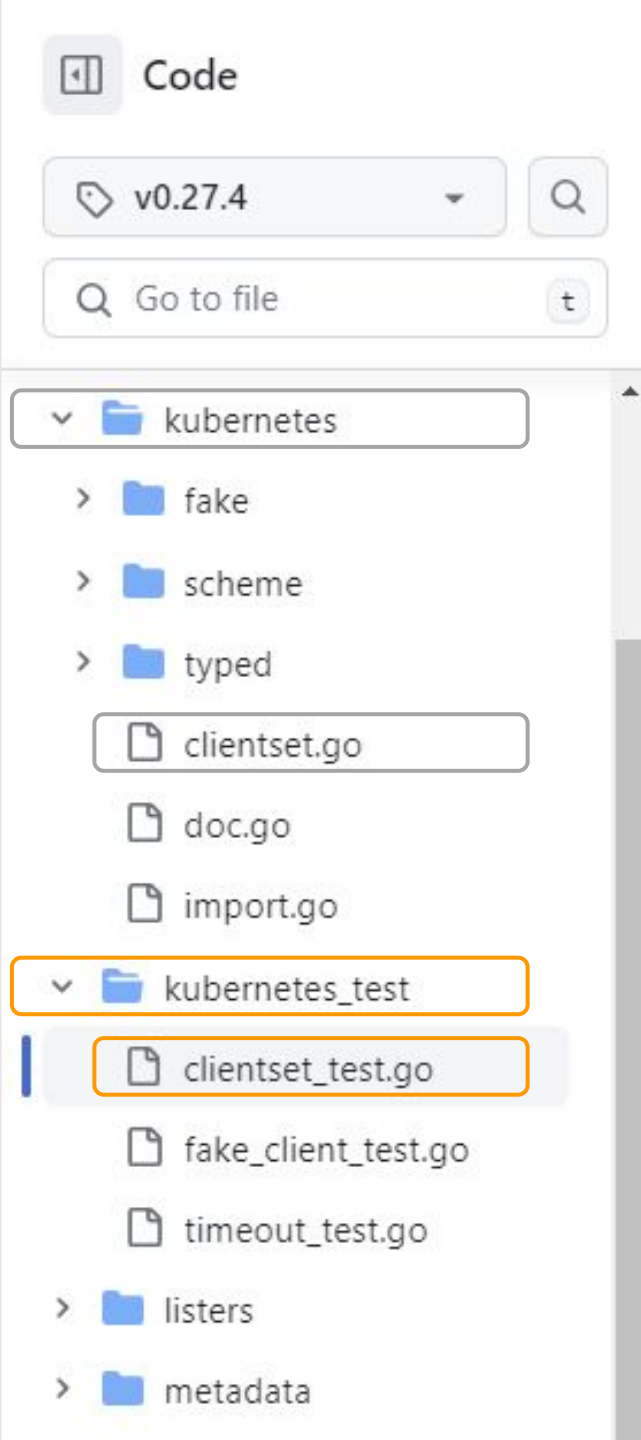
// func TestMultiply(t *testing.T) {
//     got := mysquare.multiply(-2, -2)
//     assert.Equal(t, 4, got)
// } unexported는 테스트 불가
```

# Whitebox & Blackbox #3

k8s/client-go 사례

별도 디렉토리에서  
블랙박스 테스트

[https://github.com/kubernetes/client-go/blob/v0.27.4/kubernetes\\_test/clientset\\_test.go](https://github.com/kubernetes/client-go/blob/v0.27.4/kubernetes_test/clientset_test.go)





## 4. 서브테스트 & 테이블 드리븐

- 서브테스트 - `t.Run()`을 이용한 테스트 내의 테스트
- 테이블 드리븐 - 여러 테스트케이스들을 간결하게 작성하기 위한 서브테스트 스타일 (슬라이스 vs 맵)



## Sub-test #1 테스트 안에 테스트, t.Run() 사용

```
// myabs7_test.go
func TestAbs_2depth(t *testing.T) {
    t.Run("foo", func(t *testing.T) {
        assert.Equal(t, 1, Abs(-1))
    })
    t.Run("bar", func(t *testing.T) {
        assert.Equal(t, 2, Abs(-2))
    })
}
```

내부에 t 대신 tt, subt 쓰기도 하는데  
그냥 t로 써도 된다.

```
# go test -v -run TestAbs_2depth
=== RUN    TestAbs_2depth
=== RUN    TestAbs_2depth/foo
=== RUN    TestAbs_2depth/bar
--- PASS: TestAbs_2depth (0.00s)
    --- PASS: TestAbs_2depth/foo (0.00s)
    --- PASS: TestAbs_2depth/bar (0.00s)
PASS
...
```

들여쓰기된다.  
'/이름'이 붙는다.

## Sub-test #2 서브테스트 안에 서브테스트

```
// myabs7_test.go
func TestAbs_3depth(t *testing.T) {
    t.Run("foo", func(t *testing.T) {
        t.Run("aa", func(t *testing.T) {
            assert.Equal(t, 11, Abs(-11))
        })
        t.Run("bb", func(t *testing.T) {
            assert.Equal(t, 12, Abs(-12))
        })
    })
    t.Run("bar", func(t *testing.T) {
        assert.Equal(t, 2, Abs(-2))
    })
}
```

```
# go test -v -run TestAbs_3depth
=== RUN    TestAbs_3depth
=== RUN    TestAbs_3depth/foo
=== RUN    TestAbs_3depth/foo/#00
=== RUN    TestAbs_3depth/foo/#01
=== RUN    TestAbs_3depth/bar
--- PASS: TestAbs_3depth (0.00s)
    --- PASS: TestAbs_3depth/foo (0.00s)
        --- PASS: TestAbs_3depth/foo/aa (0.00s)
        --- PASS: TestAbs_3depth/foo/bb (0.00s)
    --- PASS: TestAbs_3depth/bar (0.00s)
```

...

nested → 더 들여쓰기, /이름

# Table-driven #1.1 자료형: 구조체 슬라이스

```
// myabs6_test.go
func TestAbs_slice(t *testing.T) {
    testCases := []struct { 구조체 슬라이스
                           name string      name 필드
                           num  int
                           want int
    }{
        {"Abs(1)", 1, 9999},
        {"Abs(2)", 2, 9999},
        {"Abs(3)", 3, 9999},
    }
    for _, tc := range testCases {
        t.Run(tc.name, func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        }) 테스트케이스 → 서브테스트
    }
}
```

```
# go test -run TestAbs_slice
--- FAIL: TestAbs_slice (0.00s)
    --- FAIL: TestAbs_slice/Abs(1) (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 1
        Test:                ...
    --- FAIL: TestAbs_slice/Abs(2) (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 2
        Test:                ...
    --- FAIL: TestAbs_slice/Abs(3) (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 3
        Test:                ...
    ...
```

# Table-driven #1.2 자료형: 구조체 맵

```
// myabs6_test.go
func TestAbs_map(t *testing.T) {
    testCases := map[string]struct {
        num    int
        want    int
    }{
        "Abs(1)": {1, 9999},
        "Abs(2)": {2, 9999},
        "Abs(3)": {3, 9999},
    }
    for name, tc := range testCases {
        t.Run(name, func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

구조체 맵  
key → 이름

```
# go test -run TestAbs_map
--- FAIL: TestAbs_map (0.00s)
    --- FAIL: TestAbs_map/Abs(1) (0.00s)
    ...
        Error:          Not equal:
                        expected: 9999
                        actual  : 1
        Test:           ...
    --- FAIL: TestAbs_map/Abs(2) (0.00s)
    ...
        Error:          Not equal:
                        expected: 9999
                        actual  : 2
        Test:           ...
    --- FAIL: TestAbs_map/Abs(3) (0.00s)
    ...
        Error:          Not equal:
                        expected: 9999
                        actual  : 3
        Test:           ...
    ...
```



## Table-driven #2.1 이름: “” 하드코딩

```
// myabs6_test.go
func TestAbs_style1(t *testing.T) {
    testCases := []struct {
        num    int
        want   int
    }{
        {1, 9999},
        {2, 9999},
        {3, 9999},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

```
# go test -run TestAbs_style1
--- FAIL: TestAbs_style1 (0.00s)
    --- FAIL: TestAbs_style1/#00 (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 1
        Test:                ...
    --- FAIL: TestAbs_style1/#01 (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 2
        Test:                ...
    --- FAIL: TestAbs_style1/#02 (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 3
        Test:                ...
```

빈 문자열이면 #00, #01, #02, ...

## Table-driven #2.2 이름: “hello” 하드코딩

```
// myabs6_test.go
func TestAbs_style2(t *testing.T) {
    testCases := []struct {
        num    int
        want   int
    }{
        {1, 9999},
        {2, 9999},
        {3, 9999},
    }
    for _, tc := range testCases {
        t.Run("hello", func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

```
# go test -run TestAbs_style2
--- FAIL: TestAbs_style1 (0.00s)
    --- FAIL: TestAbs_style1/hello (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 1
        Test:                ...
    --- FAIL: TestAbs_style1/hello#01 (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 2
        Test:                ...
    --- FAIL: TestAbs_style1/hello#02 (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 3
        Test:                ...
```

같은 이름 있으면 두번째부터 번호

## Table-driven #2.3 같은 이름 있음

```
// myabs6_test.go
func TestAbs_style3(t *testing.T) {
    testCases := []struct {
        name string
        num  int
        want int
    }{
        {"negative", -1, 9999},
        {"non-negative", 0, 9999},
        {"non-negative", 1, 9999},
    }
    for _, tc := range testCases {
        t.Run(tc.name, func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

```
# go test -run TestAbs_style3
--- FAIL: TestAbs_style3 (0.00s)
    --- FAIL: TestAbs_style3/negative (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 1
        Test:                ...
    --- FAIL: TestAbs_style3/non-negative (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 0
        Test:                ...
    --- FAIL: TestAbs_style3/non-negative#01 (0.00s)
    ...
        Error:                Not equal:
                               expected: 9999
                               actual  : 1
        Test:                ...
    ...
```

같은 이름 있으면 두번째부터 번호

## Table-driven #2.4 Sprintf() 사용

```
// myabs6_test.go
func TestAbs_style4(t *testing.T) {
    testCases := []struct {
        num    int
        want    int
    }{
        {-1, 9999},
        {0, 9999},
        {1, 9999},
    }
    for i, tc := range testCases {
        t.Run(fmt.Sprintf("#%d Abs(%d)", i,
tc.num), func(t *testing.T) {
            got := Abs(tc.num)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

슬라이스 인덱스,  
입력값 활용 가능

```
# go test -run TestAbs_style4
--- FAIL: TestAbs_style4 (0.00s)
    --- FAIL: TestAbs_style4/#0_Abs(-1) (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 1
        Test:                ...
    --- FAIL: TestAbs_style4/#1_Abs(0) (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 0
        Test:                ...
    --- FAIL: TestAbs_style4/#2_Abs(1) (0.00s)
    ...
        Error:                Not equal:
                                expected: 9999
                                actual  : 1
        Test:                ...
    ...
```

공백은 '\_'로 바뀐다.



## II. Go 테스트 기능 활용

- 5. 테스트의 유형 & Benchmark
- 6. Fuzz & Example
- 7. Failfast & Skip
- 8. Parallel & race



## 5. 테스트의 유형 & Benchmark

- Test외에도 Benchmark, Fuzz, Example 테스트가 있다.
- Benchmark는 성능 측정을 위한 테스트



# 테스트의 유형

구분	Test ★★★	Benchmark ★	Fuzz	Example
함수	func TestXxx	func BenchmarkXxx	func FuzzXxx	func ExampleXxx
목적	원하는대로 작동하는지 테스트	성능 측정	특이 케이스 (잠재적 버그) 찾기	원하는대로 작동하는지 테스트 (Test와 유사하나 기능은 제한적) go doc(문서화) 연계
특징	원하는 결과와 일치하는가?  <b>결과: PASS/FAIL</b> <b>커버리지 측정 가능</b>	일정기간 반복수행하여 수행횟수 등 측정  <b>결과: 성능 지표</b>	자동 생성된 <b>랜덤값으로 테스트</b> (시드 값 지정가능) 예기치 않은 결과, panic 검출	<b>출력값 검증</b> <b>주석으로 원하는 값 입력</b> // Output: // Unordered output:

# Benchmark #1

```
// mybench.go
func RandInt() int {
    return rand.Int() 랜덤 정수
}

func Factorial(x *big.Int) *big.Int {
    n := big.NewInt(1)
    if x.Cmp(big.NewInt(0)) == 0 { 팩토리얼
        return n
    }
    return n.Mul(x, Factorial(n.Sub(x, n)))
}
```

```
// mybench_test.go
func BenchmarkRandInt(b *testing.B) {
    for i := 0; i < b.N; i++ {
        RandInt()
    }
}

func BenchmarkFactorial(b *testing.B) {
    for i := 0; i < b.N; i++ {
        Factorial(big.NewInt(0))
    }
}
```

```
# go test -bench . -benchmem
goos: linux
goarch: amd64
pkg: github.com/jmnote/go-test/mybench
cpu: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
```

BenchmarkRandInt-3	79341822	15.52 ns/op
BenchmarkFactorial-3	27896623	41.42 ns/op
PASS	이름-CPU수	실행횟수

ok	github.com/jmnote/go-test/mybench	3.456s
----	-----------------------------------	--------

0 B/op	0 allocs/op
40 B/op	2 allocs/op
메모리할당량(바이트)	할당횟수



## Benchmark #2 - benchtime 수행시간 (기본값: 1s)

```
# go test -bench . -benchtime 10s
goos: linux
goarch: amd64
pkg: github.com/jmnote/go-test/mybench
cpu: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
BenchmarkRandInt-3      821294083      14.66 ns/op
BenchmarkFactorial-3    249097959      47.87 ns/op
PASS
ok      github.com/jmnote/go-test/mybench 30.318s
```

# Benchmark #3 - count

반복수행횟수 (기본값: 1)

```
# go test -bench . -count 2
```

```
goos: linux
```

```
goarch: amd64
```

```
pkg: github.com/jmnote/go-test/mybench
```

```
cpu: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
```

BenchmarkRandInt-3	82141674	14.01 ns/op	#1
BenchmarkRandInt-3	89634460	13.64 ns/op	
BenchmarkFactorial-3	29371269	42.80 ns/op	#2
BenchmarkFactorial-3	26349254	42.63 ns/op	

```
PASS
```

```
ok      github.com/jmnote/go-test/mybench  4.890s
```

# Benchmark #4 - cpu

CPU 개수 (기본값: GOMAXPROCS)

```
# go test -bench . -cpu 2
goos: linux
goarch: amd64
pkg: github.com/jmnote/go-test/mybench
cpu: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
BenchmarkRandInt-2      77988321      13.87 ns/op
BenchmarkFactorial-2    28918156      41.77 ns/op
PASS
ok      github.com/jmnote/go-test/mybench  2.355s
```

# Benchmark #5 - 타이머 조작

```
// mybench2.go
func NewBig() *Big {
    time.Sleep(time.Second)
    return new(Big)
}

func (*Big) Len() int {
    return 42
}
```

```
// mybench2_test.go
func BenchmarkBigLen(b *testing.B) {
    big := NewBig()
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        big.Len()
    }
}
```

NewBig()도 실행하지만  
Len()만 측정하고 싶다.

만약 b.ResetTimer()가 없다면  
원치 않는 NewBig() 수행시간까지 더해진다.

```
# go test -benchmem -run=^$ -bench ^BenchmarkBigLen$ github.com/jmnote/go-test/mybench2
```

```
goos: linux
goarch: amd64
pkg: github.com/jmnote/go-test/mybench2
cpu: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
BenchmarkBigLen-3      1000000000      0.3123 ns/op      0 B/op      0 allocs/op
PASS
ok      github.com/jmnote/go-test/mybench2  6.375s
```



## 6. Fuzz & Example

- Fuzz - 랜덤값 대입 테스트 (쿼리문 검증 등에 활용)
  - Example - 출력내용 간편 검증, go doc(문서화) 연계
- ※ Fuzz는 비교적 최근 도입(k8s, prometheus에서는 gofuzz 사용)



# Fuzz #1 DontSayGoodbye

```
// myfuzz.go
func DontSayGoodbye(s string) error {
    if strings.Contains(s, "goodbye") {
        return errors.New("goodbye")
    }
    return nil
}
```

goodbye 포함되면 에러  
(실용적이지는 않지만 쉬운 코드)

```
// myfuzz_test.go
func FuzzDontSayGoodbye(f *testing.F) {
    f.Add("hello")
    f.Fuzz(func(t *testing.T, s string) {
        err := DontSayGoodbye(s)
        if err != nil {
            t.Errorf("%v", err)
        }
    })
}
```

# go test -fuzz FuzzDontSayGoodbye -fuzztime=10s 10초 동안 찾아보자.

```
fuzz: elapsed: 0s, gathering baseline coverage: 0/31 completed
fuzz: elapsed: 0s, gathering baseline coverage: 31/31 completed, now fuzzing with 3 workers
fuzz: elapsed: 3s, execs: 328809 (109602/sec), new interesting: 0 (total: 31)
fuzz: elapsed: 6s, execs: 644073 (104949/sec), new interesting: 0 (total: 31)
fuzz: elapsed: 9s, execs: 950227 (102169/sec), new interesting: 0 (total: 31)
fuzz: elapsed: 10s, execs: 1069220 (108201/sec), new interesting: 0 (total: 31)
PASS
ok      github.com/jmnote/go-test/myfuzz      10.113s
```

10초 동안 백만개의 케이스 생성하여  
테스트했는데 못찾음 → PASS

# Fuzz #2 DontSayBye

```
// myfuzz.go
func DontSayBye(s string) error {
    if strings.Contains(s, "bye") {
        return errors.New("bye")
    }
    return nil
}
```

bye 포함되면 에러

```
// myfuzz_test.go
func FuzzDontSayBye(f *testing.F) {
    f.Add("hello")
    f.Fuzz(func(t *testing.T, s string) {
        err := DontSayBye(s)
        if err != nil {
            t.Errorf("%v", err)
        }
    })
}
```

에러이면 FAIL

```
# go test -fuzz FuzzDontSayBye -fuzztime=10s
fuzz: elapsed: 0s, gathering baseline coverage: 0/85 completed
fuzz: elapsed: 0s, gathering baseline coverage: 85/85 completed, now fuzzing with 3 workers
fuzz: minimizing 102-byte failing input file
fuzz: elapsed: 1s, minimizing
```

10초 동안 찾아보자.

```
--- FAIL: FuzzDontSayBye (0.52s)
--- FAIL: FuzzDontSayBye (0.00s)
    myfuzz_test.go:27: bye
```

0.5초만에 찾음 → FAIL  
실패한 케이스를 파일에 기록

파일 내용  
go test fuzz v1  
string("bye")

Failing input written to `testdata/fuzz/FuzzDontSayBye/427e96a2173ebd9a`  
To re-run:

`go test -run=FuzzDontSayBye/427e96a2173ebd9a` 해당 케이스 다시 실행하는 방법

```
FAIL
exit status 1
```

FAIL [github.com/jmnote/go-test/myfuzz](https://github.com/jmnote/go-test/myfuzz)

0.524s

# Example #1 Output

출력결과를 간단히 테스트 (Output 주석 내용과 비교)  
- 단점: 주석이므로 변수 사용 불가

```
// myexample.go
func Hello() {
    fmt.Println("hello")
}
```

```
func HelloBye() {
    fmt.Println("hello")
    fmt.Println("bye")
}
```

```
// myexample_test.go
func ExampleHello() {
    Hello()
    // Output: hello
}
```

```
func ExampleHello_fail() {
    Hello()
    // Output: foo
}
```

```
func ExampleHelloBye() {
    HelloBye()
    // Output:
    // hello
    // bye
}
```

```
func ExampleHelloBye_fail() {
    HelloBye()
    // Output:
    // foo
    // bye
}
```

```
# go test -run ExampleHello -v
=== RUN    ExampleHello
--- PASS: ExampleHello (0.00s)
=== RUN    ExampleHello_fail
--- FAIL: ExampleHello_fail (0.00s)
got:
hello
want:
foo
=== RUN    ExampleHelloBye
--- PASS: ExampleHelloBye (0.00s)
=== RUN    ExampleHelloBye_fail
--- FAIL: ExampleHelloBye_fail (0.00s)
got:
hello
bye
want:
foo
bye
FAIL
exit status 1
```



# Example #2 Unordered output

```
// myexample.go
func Shuffle(nums []int) []int {
    rand.Shuffle(len(nums), func(i, j int) {
        nums[i], nums[j] = nums[j], nums[i]
    })
    return nums
}
```

슬라이스 뒤섞는 함수

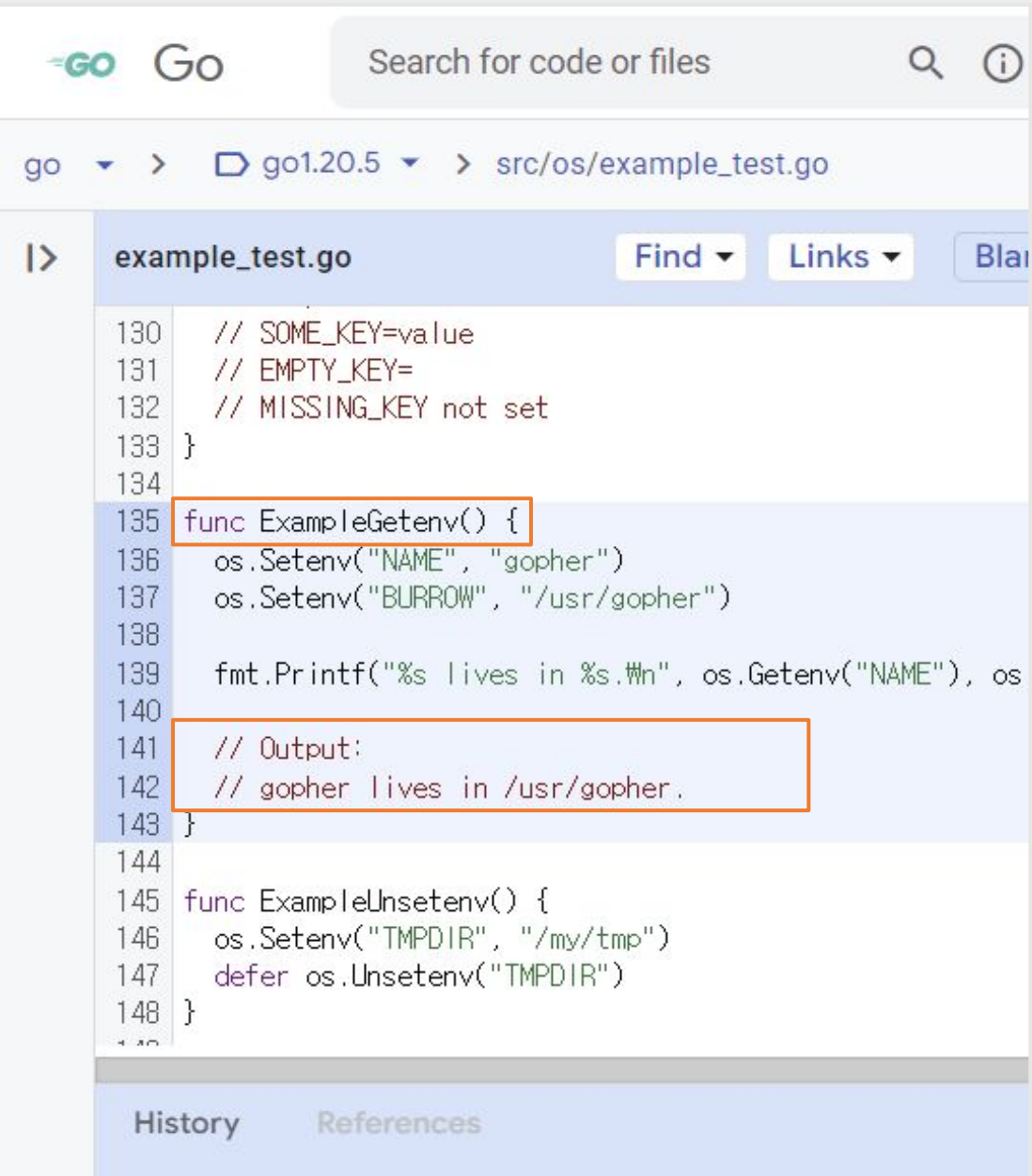
```
// myexample_test.go
func ExampleShuffle() {
    nums := Shuffle([]int{1, 2, 3, 4, 5})
    for _, value := range nums {
        fmt.Println(value)
    }
    // Unordered output:
    // 5
    // 3
    // 2
    // 4
    // 1
}
```

1~5 뒤섞어 출력  
순서 달라도  
구성이 맞으면 PASS

```
# go test -run ExampleShuffle -v
=== RUN    ExampleShuffle
--- PASS: ExampleShuffle (0.00s)
PASS
ok      github.com/jmnote/go-test/myexample    0.001s
```


# Example #3 사례

단점 있지만, 표준 라이브러리에서는 자주 사용.  
Go 문서에 Example로 등록 (+ 문서 내용 테스트)



The screenshot shows the Go source code for `example_test.go` in the `src/os` directory. The code defines a function `ExampleGetenv()` which sets environment variables and prints them. The function is highlighted with a red box. The code is as follows:

```
130 // SOME_KEY=value
131 // EMPTY_KEY=
132 // MISSING_KEY not set
133 }
134
135 func ExampleGetenv() {
136     os.Setenv("NAME", "gopher")
137     os.Setenv("BURROW", "/usr/gopher")
138
139     fmt.Printf("%s lives in %s.\n", os.Getenv("NAME"), os.Getenv("BURROW"))
140
141     // Output:
142     // gopher lives in /usr/gopher.
143 }
144
145 func ExampleUnsetenv() {
146     os.Setenv("TMPDIR", "/my/tmp")
147     defer os.Unsetenv("TMPDIR")
148 }
```



The screenshot shows the Go documentation for the `os` package. The `Getenv(key)` function is highlighted with a red box. The `Example` section is also highlighted with a red box. The example code is as follows:

```
package main

import (
    "fmt"
    "os"
)

func main() {
    os.Setenv("NAME", "gopher")
    os.Setenv("BURROW", "/usr/gopher")

    fmt.Printf("%s lives in %s.\n", os.Getenv("NAME"), os.Getenv("BURROW"))
}
```

The output of the example is shown as:

```
Output:
gopher lives in /usr/gopher.
```



# 7. Failfast & Skip

- Failfast - FAIL을 발견하면 바로 끝내자.
- Skip - 오래 걸리는 건 건너뛰자.



# [testing] Error & Fatal 함수내 중단

```
// myabs4_test.go
func TestAbs_1(t *testing.T) {
    if Abs(1) != 9999 {
        t.Error("Abs(1)")
    }
    if Abs(2) != 9999 {
        t.Error("Abs(2)")
    }
    if Abs(3) != 9999 {
        t.Error("Abs(3)")
    }
}

func TestAbs_2(t *testing.T) {
    if Abs(1) != 9999 {
        t.Fatal("Abs(1)")
    }
    if Abs(2) != 9999 {
        t.Fatal("Abs(2)")
    }
    if Abs(3) != 9999 {
        t.Fatal("Abs(3)")
    }
}
```

```
root@wsl:~/go/src/go-test/myabs4# go test
```

```
--- FAIL: TestAbs_1 (0.00s)
    myabs4_test.go:10: Abs(1)
    myabs4_test.go:13: Abs(2)
    myabs4_test.go:16: Abs(3)
--- FAIL: TestAbs_2 (0.00s)
    myabs4_test.go:22: Abs(1)
FAIL
exit status 1
FAIL    github.com/jmnote/go-test/myabs4    0.001s
```

케이스 3개 모두 실행

첫번째 케이스에서 중단  
(함수 내에서의 중단)

# [testify] assert & require 함수내 중단

```
// myabs5_test.go
```

```
import (
```

하위패키지명이 다르다

```
...
```

```
"github.com/stretchr/testify/assert"
```

```
"github.com/stretchr/testify/require"
```

```
)
```

```
func TestAbs_1(t *testing.T) {
```

```
    assert.Equal(t, 9999, Abs(1))
```

```
    assert.Equal(t, 9999, Abs(2))
```

```
    assert.Equal(t, 9999, Abs(3))
```

```
}
```

함수명은 같다

```
func TestAbs_2(t *testing.T) {
```

```
    require.Equal(t, 9999, Abs(1))
```

```
    require.Equal(t, 9999, Abs(2))
```

```
    require.Equal(t, 9999, Abs(3))
```

```
}
```

```
root@wsl:~/go/src/go-test/myabs5# go test
```

```
--- FAIL: TestAbs_1 (0.00s)
```

```
...
```

expected: 9999

actual : 1

```
...
```

expected: 9999

actual : 2

```
...
```

케이스 3개 모두 실행

expected: 9999

actual : 3

```
--- FAIL: TestAbs_2 (0.00s)
```

```
...
```

첫번째 케이스에서 중단  
(함수 내에서의 중단)

expected: 9999

actual : 1

```
FAIL
```

```
...
```

# go test --failfast 함수간 중단

[testing 예제]

```
# go test --failfast
--- FAIL: TestAbs_1 (0.00s)
    myabs4_test.go:10: Abs(1)
    myabs4_test.go:13: Abs(2)
    myabs4_test.go:16: Abs(3)
FAIL
exit status 1
FAIL    github.com/jmnote/go-test/myabs4
0.001s
```

양쪽 모두...

TestAbs\_1만 실행되고

TestAbs\_2는 실행되지 않았다.

→ 테스트 함수 중 하나라도 fail이면 중단된다.

? 여러 패키지를 테스트할 때도 중단될까?

[testify 예제]

```
# go test --failfast
--- FAIL: TestAbs_1 (0.00s)
...
Error:      Not equal:
             expected: 9999
             actual  : 1
Test:       TestAbs_1
...
Error:      Not equal:
             expected: 9999
             actual  : 2
Test:       TestAbs_1
...
Error:      Not equal:
             expected: 9999
             actual  : 3
Test:       TestAbs_1

FAIL
exit status 1
FAIL    github.com/jmnote/go-test/myabs5
0.002s
```

# 패키지간 failfast? #1 중단되지 않음

```
root@ws1:~/go/src/go-test# ls | grep my
```

myabs

myabs2

myabs3

myabs4

...

대략 하위 디렉토리(패키지) 목록...

```
root@ws1:~/go/src/go-test# go test --failfast ./... | grep 'github.com/jmnote/go-test/my'
```

ok github.com/jmnote/go-test/myabs (cached)

FAIL github.com/jmnote/go-test/myabs2 0.001s

FAIL github.com/jmnote/go-test/myabs3 0.003s

FAIL github.com/jmnote/go-test/myabs4 0.003s

...

패키지 수준 결과만 보기 위해  
grep 추가

개별 패키지 테스트는 중단되지만  
다른 패키지 테스트는 계속 진행된다.

# 패키지간 failfast? #2 대안

```
## Makefile
failfast:
    scripts/failfast.sh
```

```
## scripts/failfast.sh
#!/bin/bash
set -euo pipefail
cd $(dirname $0)/../

for package in $(go list ./...); do
    echo ===== $package =====
    echo + go test -failfast $package
    go test -failfast $package
done
```

패키지 목록에서 하나씩 뽑아서  
테스트하다가 에러 나면 중단

```
root@wsl:~/go/src/go-test# make failfast
scripts/failfast.sh
===== github.com/jmnote/go-test/myabs =====
+ go test -failfast github.com/jmnote/go-test/myabs
ok      github.com/jmnote/go-test/myabs (cached)
===== github.com/jmnote/go-test/myabs2 =====
+ go test -failfast github.com/jmnote/go-test/myabs2
--- FAIL: TestAbs (0.00s)
    myabs2_test.go:10: Abs(-2) = 2; want 2
FAIL
FAIL    github.com/jmnote/go-test/myabs2    0.001s
FAIL
make: *** [Makefile:2: failfast] Error 1
```

나쁘지 않은 방법이지만, 패키지간 순차실행만 가능.  
(반면 go test ./...로 테스트하면 여러 패키지 병렬 실행)  
스크립트에 병렬처리 추가하여 개선할 수 있겠으나,  
~~여백이 부족하여~~ 복잡하고 go와는 거리가 있어 생략...



# Skip & Short

오래 걸리는 테스트에 t.Skip을 붙여 건너뛸 수 있다.  
어떤 조건에서 건너뛸지는 구현하기 나름인데, 흔히 Short과 함께 사용한다.

```
// myskip.go
func Abs(num int) int {
    if num < 0 {
        return -num
    }
    return num
}


func SlowAbs(num int) int {
    time.Sleep(10 * time.Second)
    return Abs(num)
}
```

```
// myskip_test.go
func TestAbs(t *testing.T) {
    assert.Equal(t, 1, Abs(-1))
}

func TestSlowAbs(t *testing.T) {
    if testing.Short() {
        t.Skip("skip!!!")
    }
    assert.Equal(t, 1, SlowAbs(-1))
}
```

```
# go test -v
=== RUN    TestAbs
--- PASS: TestAbs (0.00s)
=== RUN    TestSlowAbs
--- PASS: TestSlowAbs (9.00s)
PASS
ok  github.com/jmnote/go-test/myskip 10.003s
```

```
# go test -v --short
=== RUN    TestAbs
--- PASS: TestAbs (0.00s)
=== RUN    TestSlowAbs
myskip_test.go:15: skip!!!
--- SKIP: TestSlowAbs (0.00s)
PASS
ok  github.com/jmnote/go-test/myskip 0.002s
```





## 8. Parallel & race

- Parallel 테스트 - 여러 테스트 동시 실행
  - race 테스트 - 고루틴에 대한 데이터 경합 테스트
- ※ 직접 관련은 없지만, 동시실행이라는 공통점이 있어 묶어봄;;



# 함수간 순차 실행

함수간에는 기본적으로 순차 실행이다. 그것을 확인하는 예시

```
// reverse.go
func Reverse(str string) (result string) {
    time.Sleep(time.Duration(len(str)*100) *
time.Millisecond) // 한 글자당 0.1초 지연
    for _, v := range str {
        result = string(v) + result
    }
    return
}
```

문자열을 뒤집는 함수  
길수록 오래 걸리게 함

```
// reverse_test.go
func TestReverse_1(t *testing.T) {
    got := Reverse("hello")      5글자: 0.5초
    assert.Equal(t, "olleh", got)
}

func TestReverse_2(t *testing.T) {
    got := Reverse("foo")        3글자: 0.3초
    assert.Equal(t, "oof", got)
}
```

```
# go test -v -run ^TestReverse_
=== RUN    TestReverse_1
--- PASS: TestReverse_1 (0.50s)
=== RUN    TestReverse_2
--- PASS: TestReverse_2 (0.30s)
PASS
ok      github.com/jmnote/go-test/myreverse 0.804s
```

0.5초 + 0.3초 = 총 0.8초 (순차)

# t.Parallel #1 함수간 병렬

```
// reverse_test.go
func TestReverseParallel_1(t *testing.T) {
    t.Parallel()
    got := Reverse("hello")
    assert.Equal(t, "olleh", got)
}

func TestReverseParallel_2(t *testing.T) {
    t.Parallel()
    got := Reverse("foo")
    assert.Equal(t, "oof", got)
}
```

```
# go test -v -run ^TestReverseParallel_
=== RUN    TestReverseParallel_1
=== PAUSE TestReverseParallel_1
=== RUN    TestReverseParallel_2
=== PAUSE TestReverseParallel_2
=== CONT   TestReverseParallel_1
=== CONT   TestReverseParallel_2
--- PASS: TestReverseParallel_2 (0.30s)
--- PASS: TestReverseParallel_1 (0.50s)
PASS
ok  github.com/jmnote/go-test/myreverse 0.503s
```

PAUSE 상태로  
예약 후...

병렬 실행

0.3초 < 0.5초 = 총 0.5초 (병렬)

## t.Parallel #2 테스트케이스간 병렬

```
// reverse_test.go
func TestReverseParallel(t *testing.T) {
    testCases := []struct {
        input string
        want  string
    }{
        {"hello", "olleh"},
        {"foo", "oof"},
    }
    for _, tc := range testCases {
        tc := tc // capture range variable
        t.Run(tc.input, func(t *testing.T) {
            t.Parallel()
            got := Reverse(tc.input)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

for문 내부에 tc 변수 할당 필요.  
※ 없으면? 마지막 tc 여러번 수행  
(defer처럼 예약 후 실행되는데  
range 앞의 tc는 덮어써진다.)

```
# go test -v -run ^TestReverseParallel$
=== RUN    TestReverseParallel
=== RUN    TestReverseParallel/hello
=== PAUSE TestReverseParallel/hello
=== RUN    TestReverseParallel/foo
=== PAUSE TestReverseParallel/foo
=== CONT   TestReverseParallel/hello
=== CONT   TestReverseParallel/foo
--- PASS: TestReverseParallel (0.00s)
    --- PASS: TestReverseParallel/foo (0.30s)
    --- PASS: TestReverseParallel/hello (0.50s)
PASS
ok  github.com/jmnote/go-test/myreverse 0.503s
```

0.3초 < 0.5초 = 총 0.5초 (병렬)

# 패키지간 병렬 실행?

패키지간에도 기본적으로 병렬이다.

정확히는 -p 옵션만큼 병렬이며, 그 기본값은 CPU개수

```
// mypkgs/pkg1.go
// mypkgs/pkg2.go
func Test1(t *testing.T) {
    time.Sleep(time.Second)
    t.Fail()
}

func Test2(t *testing.T) {
    time.Sleep(time.Second)
    t.Fail()
}

func Test3(t *testing.T) {
    time.Sleep(time.Second)
    t.Fail()
}
```

1초 걸리는 함수 3개

```
root@ws1:~/go/src/go-test/mypkgs# time go test ./...
--- FAIL: Test1 (1.00s)
--- FAIL: Test2 (1.00s)
--- FAIL: Test3 (1.00s)
FAIL
FAIL    github.com/jmnote/go-test/mypkgs/pkg1    3.002s
--- FAIL: Test1 (1.00s)
--- FAIL: Test2 (1.00s)
--- FAIL: Test3 (1.00s)
FAIL
FAIL    github.com/jmnote/go-test/mypkgs/pkg2    3.002s
FAIL

real    0m3.189s
user    0m0.223s
sys     0m0.055s
```

리눅스 명령어 time으로 측정

pkg1, pkg2 각각 3초인데  
→ 총 3초 (병렬)

# 패키지간 순차 실행

-p 옵션의 기본값은 GOMAXPROCS(기본값: runtime.NumCPU()).  
한편, -parallel 옵션은 -p 옵션과 유사하지만 Fuzz에만 적용된다.

```
# time go test -p 1 ./...  
--- FAIL: Test1 (1.00s)  
--- FAIL: Test2 (1.00s)  
--- FAIL: Test3 (1.00s)  
FAIL  
FAIL    github.com/jmnote/go-test/mypkgs/pkg1  
3.004s  
--- FAIL: Test1 (1.00s)  
--- FAIL: Test2 (1.00s)  
--- FAIL: Test3 (1.00s)  
FAIL  
FAIL    github.com/jmnote/go-test/mypkgs/pkg2  
3.004s  
FAIL
```

real	0m6.258s	총 6초 (순차)
user	0m0.135s	
sys	0m0.113s	

```
# time go test -parallel 1 ./...  
--- FAIL: Test1 (1.00s)  
--- FAIL: Test2 (1.00s)  
--- FAIL: Test3 (1.00s)  
FAIL  
FAIL    github.com/jmnote/go-test/mypkgs/pkg1  
3.003s  
--- FAIL: Test1 (1.00s)  
--- FAIL: Test2 (1.00s)  
--- FAIL: Test3 (1.00s)  
FAIL  
FAIL    github.com/jmnote/go-test/mypkgs/pkg2  
3.003s  
FAIL
```

real	0m3.193s	총 3초 (병렬) -parallel은 fuzz에만 적용되므로 효과 없음
user	0m0.173s	
sys	0m0.090s	

# race 테스트

데이터 경합(data race)이 있는지 점검한다.

※ 2개 이상의 고루틴이 같은 변수 동시 접근, 그중 쓰기(write)가 있는 경우

```
// myrace.go
func Count() {
    i := 0
    go func() {
        for {
            i++ // write
            time.Sleep(time.Second)
            fmt.Println("in", i)
        }
    }()
    time.Sleep(3 * time.Second)
    fmt.Println("out", i) // read
}
```

```
# go test
in 1
in 2
out 3
PASS
ok      github.com/jmnote/go-test/myrace  3.003s
```

그냥 테스트하면 PASS

```
# go test -race
in 1
in 2
=====
WARNING: DATA RACE
Read at 0x00c000134148 by goroutine 7:
  github.com/jmnote/go-test/myrace.Count()
    /root/go/src/go-test/myrace/myrace.go:18 +0xe6
  github.com/jmnote/go-test/myrace.TestCount()
    /root/go/src/go-test/myrace/myrace_test.go:8
+0x24
...
FAIL    github.com/jmnote/go-test/myrace  3.008s
```

race 테스트하면 FAIL  
(고루틴을 사용한다면 해보자)





## III. 테스트 사례

- 9. 변화하는 것에 대한 테스트
- 10. Error & Panic 테스트
- 11. HTTP 테스트



## 9. 변화하는 것에 대한 테스트

랜덤 같이 일관되지 않은 결과를 테스트하는 경우



# 주사위 던지기

```
// myrand.go
func Dice() int {
    return rand.Intn(6) + 1
}
```

1~6 사이의 랜덤 정수 반환

```
// myrand_test.go
func TestDice(t *testing.T) {
    for i := 0; i < 10000; i++ {
        got := Dice()
        assert.Contains(t, []int{1, 2, 3, 4, 5, 6}, got)
    }
}
```

10000번 수행하여  
각각 1~6 사이의  
정수인지 확인

```
// myrand_test.go
func TestDice_2(t *testing.T) {
    for i := 0; i < 10000; i++ {
        got := Dice()
        assert.GreaterOrEqual(t, got, 1)
        assert.LessOrEqual(t, got, 6)
    }
}
```

10000번 수행하여  
각각 1이상, 6이하인지 확인

```
func TestDice_3(t *testing.T) {
    min := 9999
    max := -9999
    for i := 0; i < 10000; i++ {
        got := Dice()
        if got > max {
            max = got
        }
        if got < min {
            min = got
        }
    }
    assert.Equal(t, 1, min)
    assert.Equal(t, 6, max)
}
```

10000번 수행하여  
최소값이 1이고  
최대값이 6인지 확인

# Logger 시간에 따라 결과가 다른 예시

```
// mylogrus.go
func LogString(str string) string {
    var buffer bytes.Buffer
    logger := logrus.New()
    logger.Out = &buffer
    logger.Info(str)
    return buffer.String()
}
```

로거 출력을 문자열로 반환

```
// mylogrus_test.go
func TestLogString(t *testing.T) {
    got := LogString("Hello")
    t.Log(got)
    assert.Contains(t, got, "level=info msg=Hello")
    assert.Regexp(t, `time="[^"]+" level=info msg=Hello`, got)
}
```

부분문자열만 확인하거나  
정규표현식으로 확인

```
root@wsl:~/go/src/go-test/mylogrus# go test -v
```

```
=== RUN    TestLogHello
```

```
mylogrus_test.go:11: time="2023-07-02T23:18:47+09:00" level=info msg=Hello
```

```
--- PASS: TestLogHello (0.00s)
```

```
PASS
```

```
ok
```

```
github.com/jmnote/go-test/mylogrus
```

```
0.002s
```

얻은 값(got)의 내용

# myclock

시간아 멈추어다오 → 별도의 시계 만들어 사용

<https://github.com/kuoss/lethe/blob/v0.2.1/storage/fileservice/delete.go#L19>

```
// clock.go
package clock

import (
    "time"
)

var (
    playgroundMode = false
    playgroundTime = time.Date(2009, 11, 10, 23,
0, 0, 0, time.UTC) // Go Playground - time.Now()
)

func Now() time.Time {
    if playgroundMode {
        return playgroundTime
    }
    return time.Now()
}

func SetPlaygroundMode(newPlaygroundMode bool) {
    playgroundMode = newPlaygroundMode
}
```

```
// delete.go
func (s *FileService) DeleteByAge() error {
    point :=
strings.Replace(clock.Now().Add(-retentionTi
me).UTC().String()[0:13], " ", "_", 1)
    files, err := s.ListFiles()
```

... 보존기간이 지난 파일을 삭제하는 함수  
→ time.Now() 대신 clock.Now() 사용

- 테스트데이터를 playgroundTime를 기준으로 작성해두고,
- 해당 데이터를 사용하여 테스트를 할 때는 SetPlaygroundMode(true)

※ Go Playground에서도 TestXxx 실행 가능

※ k8s의 fakeClock 사례

[https://github.com/kubernetes/kubernetes/blob/v1.27.4/pkg/kubelet/kubelet\\_test.go#L136](https://github.com/kubernetes/kubernetes/blob/v1.27.4/pkg/kubelet/kubelet_test.go#L136)



# 10. Error & Panic 테스트

- Error를 테스트하자
- Panic을 테스트하자



# Error 테스트 #1

ok/error 케이스를 각각 함수로

```
// myerror.go
func ToInt(str string) (int, error) {
    num, err := strconv.Atoi(str)
    if err != nil {
        return 0, errors.New("not int")
    }
    return num, nil
}
```

정수 문자열을 정수로 바꾸는 함수

```
// myerror_test.go
func TestToInt_ok(t *testing.T) {
    testCases := []struct {
        input string
        want int
    }{
        {"11", 11},
        {"-2", -2},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := ToInt(tc.input)
            assert.NoError(t, err)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

에러가 없었고  
원하는 값이 나왔는가?

```
// myerror_test.go
func TestToInt_error(t *testing.T) {
    testCases := []struct {
        input string
        want int
        wantError string
    }{
        {"", 0, "not int"},
        {"a", 0, "not int"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := ToInt(tc.input)
            assert.EqualError(t, err, tc.wantError)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

원하는 에러와 값이 나왔는가?

# Error 테스트 #2

ok/error 케이스를  
함수 하나에  
→ 서브 테스트 활용

```
// myerror2_test.go
func TestToInt(t *testing.T) {
    t.Run("ok", func(t *testing.T) {
        testCases := []struct {
            input string
            want int
        }{
            {"11", 11},
            {"-2", -2},
        }
        for _, tc := range testCases {
            ...
        }
    })
    t.Run("error", func(t *testing.T) {
        testCases := []struct {
            input string
            want int
            wantError string
        }{
            {"", 0, "not int"},
            {"a", 0, "not int"},
        }
        for _, tc := range testCases {
            ...
        }
    })
}
```



# Error 테스트 #3

ok/error 케이스를  
함수 하나에 v2  
→ if wantError

```
// myerror3_test.go
func TestToInt(t *testing.T) {
    testCases := []struct {
        input      string
        want        int
        wantError   string
    }{
        {"11", 11, ""},
        {"-2", -2, ""},
        {"", 0, "not int"},
        {"a", 0, "not int"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := ToInt(tc.input)
            if tc.wantError == "" {
                assert.NoError(t, err)
            } else {
                assert.EqualError(t, err, tc.wantError)
            }
            assert.Equal(t, tc.want, got)
        })
    }
}
```

원하는 에러가 없으면 NoError 검사  
원하는 에러가 있으면 EqualError 검사  
(공통) 원하는 값이 나왔는가?

# Panic 테스트

```
// mypanic.go
func ToInt(str string) (v int, err error) {
    defer myRecover(str, &err)
    v = toInt(str)
    return v, nil
}

func toInt(str string) int {
    v, err := strconv.Atoi(str)
    if err != nil {
        panic(errors.New("not int"))
    }
    return v
}

func myRecover(str string, errp *error) {
    e := recover()
    if e == nil {
        return
    }
    *errp = fmt.Errorf("%v", e)
}
```

panic/recover 활용 에러처리 패턴

```
// mypanic_test.go
func Test_toInt(t *testing.T) {
    testCases := []struct {
        str          string
        want          int
        wantPanic     string
    }{
        {"11", 11, ""},
        {"-2", -2, ""},
        {"", 0, "not int"},
        {"a", 0, "not int"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            if tc.wantPanic == "" {
                got := toInt(tc.str)
                assert.Equal(t, tc.want, got)
            } else {
                assert.PanicsWithError(t, tc.wantPanic, func() {
                    toInt(tc.str)
                })
            }
        })
    }
}
```

원하는 패닉 없으면 값만 검사

원하는 패닉 있으면 PanicsWithError 검사

- 자체 recover 기능 있다.
- 패닉이 일어나지 않아도 FAIL이다.



# 11. HTTP 테스트

- httptest 패키지의 Recorder를 사용하여 HTTP 서버 기능을 테스트해보자.



# HTTP 테스트 #1

Recorder 사용하면 웹서버 구동 없이 테스트 가능

```
// myhttp.go
import (
    "github.com/gin-gonic/gin"
)

func setupRouter() *gin.Engine {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.String(200, "pong")
    })
    return r    ping 요청에 200 pong
}

func Run() {
    r := setupRouter()
    r.Run(":8080")
}
```

```
// myhttp_test.go
var router = setupRouter()

func TestPing_1(t *testing.T) {
    w := httptest.NewRecorder()    Recorder 사용
    req, err := http.NewRequest("GET", "/ping", nil)
    assert.NoError(t, err)        http GET 요청
    router.ServeHTTP(w, req)
    assert.Equal(t, 200, w.Code)
    assert.Equal(t, "pong", w.Body.String())
}
```

# HTTP 테스트 #2

httptest에도 NewRequest가 있다

```
// myhttp.go
import (
    "github.com/gin-gonic/gin"
)

func setupRouter() *gin.Engine {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.String(200, "pong")
    })
    return r
}

func Run() {
    r := setupRouter()
    r.Run(":8080")
}
```

Run()은 테스트 안하나요?  
→ '17. 커버리지 100%' 참고

```
// myhttp_test.go
var router = setupRouter()

func TestPing_2(t *testing.T) {
    w := httptest.NewRecorder()
    req := httptest.NewRequest("GET", "/ping", nil)
    router.ServeHTTP(w, req)      httptest GET 요청
    assert.Equal(t, 200, w.Code)  (err 없어서 편안)
    assert.Equal(t, "pong", w.Body.String())
}
```

# HTTP 테스트 #3

```
// myhttp2.go
func setupRouter() *gin.Engine {
    router := gin.Default()
    v1.GET("/v1/login", loginEndpoint)
    v1.GET("/v1/read", readEndpoint)
    return router
}

func loginEndpoint(c *gin.Context) {
    c.String(200, "pong")
}

func readEndpoint(c *gin.Context) {
    c.String(200, "pong")
}
```

```
// myhttp2_test.go
func TestRouter(t *testing.T) {
    router := setupRouter()
    testCases := []struct { 테스트케이스
        method    string
        path       string
        wantCode   int
        wantBody   string
    }{
        {
            "POST", "/v1/write",
            404, "404 page not found",
        },
        {
            "GET", "/v1/read",
            200, "pong",
        },
        ...
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            w := httptest.NewRecorder()
            req := httptest.NewRequest(tc.method,
tc.path, nil)
            router.ServeHTTP(w, req)
            assert.Equal(t, tc.wantCode, w.Code)
            assert.Equal(t, tc.wantBody, w.Body.String())
        })
    }
    ...
}
```

# HTTP 테스트 #4

```
// myhttp4.go
func setupRouter() *gin.Engine {
    r := gin.Default()
    r.POST("/api/login", apiLogin)
    return r
}

func apiLogin(c *gin.Context) {
    if c.PostForm("username") != "hello" ||
    c.PostForm("password") != "world" {
        c.JSON(403, gin.H{"status": "error",
            "error": "username or password is incorrect"})
        return 로그인 실패하면 403
    }
    c.JSON(200, gin.H{"status": "ok", "data":
    gin.H{"token": "EXAMPLE", "message": "logged in
    successfully"}}) 성공하면 200 + 토큰
}
```

```
// myhttp4_test.go
func TestAPILogin(t *testing.T) {
    testCase := []struct {
        username string
        password string
        wantCode int
        wantBody string
    }{
        {
            "hello", "world",
            200, `{"data":{"message":"logged
in successfully", "token":"EXAMPLE"},
"status":"ok"}`,
        }, ...
    }
    for _, tc := range testCase {
        t.Run("", func(t *testing.T) {
            w := httptest.NewRecorder()
            ...
            assert.Equal(t, tc.wantCode,
w.Code)
            assert.JSONEq(t, tc.wantBody,
w.Body.String())
        })
    }
}
```

# HTTP 테스트 #5

## rate limit 테스트 #1

```
// myhttp5.go
import (
    "github.com/gin-gonic/gin"
    "go.uber.org/ratelimit"
)

// 초당 3회 → 대략 0.333초 지연
var limit = ratelimit.New(3)

func rateLimit() gin.HandlerFunc {
    return func(ctx *gin.Context) {
        limit.Take()
    }
}

func setupRouter() *gin.Engine {
    r := gin.Default()
    api := r.Group("/api")
    api.Use(rateLimit())
    api.GET("/ping", apiPing)
    return r
}

// /api 그룹에 적용

func apiPing(c *gin.Context) {
    c.String(200, "pong")
}
```

```
// myhttp5_test.go
func getAPIPing() (int, string) {
    w := httptest.NewRecorder()
    req := httptest.NewRequest("GET", "/api/ping", nil)
    router.ServeHTTP(w, req)
    return w.Code, w.Body.String()
}
```

```
func TestRateLimit(t *testing.T) {
    for i := 0; i < 5; i++ {
        start := time.Now()
        code, body := getAPIPing()
        elapsed := time.Since(start)
        assert.Equal(t, 200, code)
        assert.Equal(t, "pong", body)
        if i > 0 {
            assert.Greater(t, elapsed,
                time.Duration(300*time.Millisecond)) 소요시간 > 300ms?
        }
    }
}
```

```
root@wsl:~/go/src/go-test/myhttp5# go test -run TestRateLimit
```

[GIN]	2023/05/05	- 02:53:01	200	3.6µs	192.0.2.1	GET "/api/ping"
[GIN]	2023/05/05	- 02:53:01	200	333.6ms	192.0.2.1	GET "/api/ping"
[GIN]	2023/05/05	- 02:53:01	200	333.8ms	192.0.2.1	GET "/api/ping"
[GIN]	2023/05/05	- 02:53:02	200	332.7ms	192.0.2.1	GET "/api/ping"
[GIN]	2023/05/05	- 02:53:02	200	333.8ms	192.0.2.1	GET "/api/ping"

2번째부터 333ms 지연

PASS

ok

github.com/jmnote/go-test/myhttp5

1.341s



# HTTP 테스트 #6

```
// myhttp6.go           rate limit 테스트 #2
import "golang.org/x/time/rate"
```

```
var limit =
rate.NewLimiter(rate.Every(time.Second),
3)
```

```
func rateLimit() gin.HandlerFunc {
    return func(c *gin.Context) {
        if !limit.Allow() {
            c.String(429, "Too Many
Requests")
            c.Abort()
        }
    }
}
```

```
func setupRouter() *gin.Engine {
    r := gin.Default()
    api := r.Group("/api")
    api.Use(rateLimit())
    api.GET("/ping", apiPing)
    return r
}
```

...

```
// myhttp6_test.go
func getAPIPing() (int, string) {
    w := httptest.NewRecorder()
    req := httptest.NewRequest("GET", "/api/ping", nil)
    router.ServeHTTP(w, req)
    return w.Code, w.Body.String()
}
```

```
func TestRateLimit(t *testing.T) {
    for i := 0; i < 5; i++ {
        code, body := getAPIPing()
        if i < 3 {
            assert.Equal(t, 200, code)
            assert.Equal(t, "pong", body)
        } else {
            assert.Equal(t, 429, code)
            assert.Equal(t, "Too Many Requests", body)
        }
    }
}
```

```
root@wsl:~/go/src/go-test/myhttp5# go test -run TestRateLimit
```

```
...
[GIN] 2023/05/05 - 03:22:19 | 200 | 3.642µs | 192.0.2.1 | GET "/api/ping"
[GIN] 2023/05/05 - 03:22:19 | 200 | 1.083µs | 192.0.2.1 | GET "/api/ping"
[GIN] 2023/05/05 - 03:22:19 | 200 | 671ns | 192.0.2.1 | GET "/api/ping"
[GIN] 2023/05/05 - 03:22:19 | 429 | 1.083µs | 192.0.2.1 | GET "/api/ping"
[GIN] 2023/05/05 - 03:22:19 | 429 | 1.083µs | 192.0.2.1 | GET "/api/ping"
PASS
```

PASS

ok

github.com/jmnote/go-test/myhttp5

1.340s



## IV. mock & 수명주기

- 12. mock
- 13. HTTP서버 mock
- 14. TestMain & suite



## 12. mock

- mock을 만들자.
  - testify/mock과 mockery를 사용해보자.
- ※ 여기서는 다루지 않았으나, gomock도 있다.



# mock #1 mock 없음

테스트 대상

```
// mymock.go
func Get(name string) (string, error) {
    code, body := catclient.HTTPGet(name)
    if code != 200 {
        return "", fmt.Errorf("err: %s",
body)
    }
    return body, nil
}
```

mocking 대상

```
// ./catclient/catclient.go
var count = 0

func HTTPGet(name string) (int, string) {
    count++
    switch count {
    case 1:
        return 200, "Hello " + name
    case 2:
        return 200, "Bye " + name
    }
    return 403, "limit exceeded"
}
```

테스트 코드

```
// mymock_test.go
func TestGet(t *testing.T) {
    testCases := []struct {
        name        string
        want        string
        wantError    string
    }{
        {"Alice", "Hello Alice", ""},
        {"Bob", "Bye Bob", ""},
        {"Carol", "", "err: limit exceeded"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := Get(tc.name)
            if tc.wantError == "" {
                assert.NoError(t, err)
            } else {
                assert.EqualError(t, err, tc.wantError)
            }
            assert.Equal(t, tc.want, got)
        })
    }
}
```

## mock #2 함수 mock

```
// mymock2.go
var httpGet = catclient.HTTPGet 함수 변수화

func Get(name string) (string, error) {
    code, body := httpGet(name)
    if code != 200 {
        return "", fmt.Errorf("err: %s", body)
    }
}
```

```
// ./catclient/catclient.go
var count = 0

func HTTPGet(name string) (int, string) {
    count++
    switch count {
    case 1:
        return 200, "Hello " + name
    case 2:
        return 200, "Bye " + name
    }
    return 403, "limit exceeded"
}
```

```
// mymock2_test.go
func mockHTTPGet(name string) (int, string) {
    switch name {
    case "Alice":
        return 200, "Hello Alice"
    case "Bob":
        return 200, "Bye Bob"
    case "Carol":
        return 403, "limit exceeded"
    }
    return 0, ""
}
```

함수 mock 작성

```
func TestGet(t *testing.T) {
    httpGet = mockHTTPGet mock으로 교체
    testCases := []struct {
        name        string
        want        string
        wantError    string
    }{
        {"Alice", "Hello Alice", ""},
        {"Bob", "Bye Bob", ""},
        {"Carol", "", "err: limit exceeded"},
    }
    ...
}
```

## mock #3 mock 없음

```
// mymock3.go
var catClient = catclient.NewCatClient()

func Get(name string) (string, error) {
    code, body := catClient.HTTPGet(name)
    if code != 200 {
        return "", fmt.Errorf("err: %s", bod
```

```
// ./catclient/catclient.go
type CatClient struct {
    count int
}
```

```
func NewCatClient() *CatClient {
    return &CatClient{}
}
```

```
func (c *CatClient) HTTPGet(name string) (in
    c.count++
    switch c.count {
    case 1:
        return 200, "Hello " + name
    case 2:
        return 200, "Bye " + name
    ...
```

```
// mymock3_test.go
```

```
func TestGet(t *testing.T) {
```

```
    testCases := []struct {
```

```
        name      string
```

```
        want      string
```

```
        wantError string
```

```
    }{
```

```
        {"Alice", "Hello Alice", ""},
```

```
        {"Bob", "Bye Bob", ""},
```

```
        {"Carol", "", "err: limit exceeded"},
```

```
    }
```

```
    for _, tc := range testCases {
```

```
        t.Run("", func(t *testing.T) {
```

```
            got, err := Get(tc.name)
```

```
            if tc.wantError == "" {
```

```
                assert.NoError(t, err)
```

```
            } else {
```

```
                assert.EqualError(t, err, tc.wantError)
```

```
            }
```

```
            assert.Equal(t, tc.want, got)
```

```
        })
```

```
    }
```

```
}
```

## mock #4 타입 mock

```
// mymock4.go
var catClient = catclient.NewCatClient()

func Get(name string) (string, error) {
    code, body := catClient.HTTPGet(name)
    if code != 200 {
        return "", fmt.Errorf("err: %s", body)
    }
}
```

```
// ./catclient/catclient.go
type ICatClient interface {
    HTTPGet(name string) (int, string)
}
```

인터페이스 추가

```
type CatClient struct {
    count int
}
```

```
func NewCatClient() ICatClient {
    return &CatClient{}
}
```

인터페이스 적용

```
func (c *CatClient) HTTPGet(name string) (int, string) {
    ...
}
```

```
// mymock4_test.go
```

```
type mockCatClient struct{}
```

```
func newMockCatClient() catclient.ICatClient {
    return &mockCatClient{}
}
```

```
func (m *mockCatClient) HTTPGet(name string) (int, string) {
    switch name {
    case "Alice":
        return 200, "Hello Alice"
    case "Bob":
        return 200, "Bye Bob"
    case "Carol":
        return 403, "limit exceeded"
    }
    return 0, ""
}
```

mock 작성

```
func TestGet(t *testing.T) {
    catClient = newMockCatClient() mock으로 교체
    testCases := []struct {
        ...
    }
}
```

# mock #5 testify/mock

```
// mymock5.go
var catClient = catclient.NewCatClient()

func Get(name string) (string, error) {
    code, body := catClient.HTTPGet(name)
    if code != 200 {
        return "", fmt.Errorf("error")
    }
}
```

```
// ./catclient/catclient.go
type ICatClient interface {
    HTTPGet(name string) (int, string)
}

type CatClient struct {
    count int
}

func NewCatClient() ICatClient {
    return &CatClient{}
}

func (c *CatClient) HTTPGet(name string) (int, string) {
    ...
}
```

```
// mymock5_test.go
```

```
import (
```

```
...
```

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

```
)
```

```
type mockCatClient struct{ mock.Mock }
```

mock 작성

```
func newMockCatClient() *mockCatClient { return &mockCatClient{}
}
```

```
func (m *mockCatClient) HTTPGet(name string) (int, string) {
    args := m.Called(name)
    return args.Int(0), args.String(1)
}
```

```
func TestGet(t *testing.T) {
    m := newMockCatClient()
    m.On("HTTPGet", "Alice").Return(200, "Hello Alice").Once()
    m.On("HTTPGet", "Bob").Return(200, "Bye Bob").Once()
    m.On("HTTPGet", "Carol").Return(403, "limit exceeded")
}
```

```
catClient = m
```

```
testCases := []struct {
```

```
...
```



# mock #6.1 mockery - mock 코드 생성 도구 (testify/mock 기반)

## 설치

```
root@wsl:~# go install github.com/vektra/mockery/v2@latest
...
root@wsl:~# mockery --version
...
v2.31.1
```

실행이 안되면 GOPATH/bin을 PATH에 추가해보자.  
(v3.0.0-alpha.0 나왔으나, 여기서는 v2를 사용하였다.)

## 실행

```
root@wsl:~/go/src/go-test/mymock6# mockery --name ICatClient
...
09 Jul 23 17:22 KST INF writing mock to file dry-run=false interface=ICatClient
qualified-name=github.com/jmnote/go-test/mymock6 version=v2.31.1
```

## 결과

```
root@wsl:~/go/src/go-test/mymock6# cat mocks/ICatClient.go
// Code generated by mockery v2.31.1. DO NOT EDIT.

package mocks

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

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

mocks 디렉토리(패키지)와 파일이 생성되었다.

## mock #6.2 mockery

```
// mymock6.go
var catClient = catclient.NewCatClient()

func Get(name string) (string, error) {
    code, body := catClient.HTTPGet(name)
    if code != 200 {
        return "", fmt.Errorf("error")
    }
}
```

```
// ./catclient/catclient.go
type ICatClient interface {
    HTTPGet(name string) (int, string)
}

type CatClient struct {
    count int
}

func NewCatClient() ICatClient {
    return &CatClient{}
}

func (c *CatClient) HTTPGet(name string) (int, string) {
    ...
}
```

```
// mymock6_test.go
```

```
import (
```

```
...
```

```
"github.com/jmnote/go-test/mymock6/mocks"
```

```
)

func TestGet(t *testing.T) {
```

```
    m := mocks.NewICatClient(t)
```

mock 작성

```
    m.On("HTTPGet", "Alice").Return(200, "Hello Alice").Once()
```

```
    m.On("HTTPGet", "Bob").Return(200, "Bye Bob").Once()
```

```
    m.On("HTTPGet", "Carol").Return(403, "limit exceeded")
```

```
    catClient = m
```

```
    testCases := []struct {
```

```
    ...
}
```

생성된 mocks가 형식을 맞춰주어 코드가 간결해졌다.

한편, 특별한 기능이 있는 mock이 필요하다면,  
라이브러리를 찾아보자. (예: go-sqlmock, httptest)



# 13. HTTP서버 mock

httptest 패키지를 가지고, 테스트 서버(HTTP 서버 mock)를 만들어보자.

※ 테스트서버의 호스트는 127.0.0.1이며, 포트는 시스템에 의해 임의할당된다.  
방법이 전혀 없는 것은 아니지만, 일반적으로 특정 포트를 지정할 수 없다.  
(특정 포트를 지정할 경우, 병렬 테스트에서 포트 충돌이 발생할 수 있다.)



# HTTP서버 mock #1

외부 API(/api/{name} → “Hello {name}” 응답)에 대한 mock

```
// myhttpmock.go
var endpoint = "http://example.com/api"

func Get(name string) (string, error) {
    resp, err := http.Get(endpoint + "/" +
name)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()
    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }
    if resp.StatusCode != 200 {
        return "", fmt.Errorf("err: %s", body)
    }
    return string(body), nil
}
```

외부 API 이용하는 함수

```
// myhttpmock_test.go
func TestGet(t *testing.T) {
    ts := httptest.NewServer(http.HandlerFunc(func(w
http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "Hello Alice")
    }))
    defer ts.Close()
    endpoint = ts.URL
    got, err := Get("Alice")
    assert.NoError(t, err)
    assert.Equal(t, "Hello Alice", got)
}
```

무조건 Hello Alice 응답

엔드포인트를 테스트서버로 교체

# HTTP서버 mock #2

mock 서버에 라우팅 구현

```
// myhttpmock2.go
var endpoint = "http://ex

func Get(name string) (string, error) {
    resp, err := http.Get(name)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()
    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }
    if resp.StatusCode != 200 {
        return "", fmt.Errorf("status code: %d", resp.StatusCode)
    }
    return string(body), nil
}
```

```
// myhttpmock2_test.go
func TestGet(t *testing.T) {
    mux := http.NewServeMux()
    mux.HandleFunc("/Alice", func(w http.ResponseWriter, r *http.Request) {
        io.WriteString(w, "Hello Alice")
    })
    mux.HandleFunc("/Bob", func(w http.ResponseWriter, r *http.Request) {
        io.WriteString(w, "Hello Bob")
    })
    ts := httptest.NewUnstartedServer(mux)
    ts.Start()
    defer ts.Close()
    endpoint = ts.URL

    got, err := Get("Alice")
    assert.NoError(t, err)
    assert.Equal(t, "Hello Alice", got)
    got, err := Get("Bob")
    assert.NoError(t, err)
    assert.Equal(t, "Hello Bob", got)
    ...
}
```

/Alice, /Bob에 응답

고급 기능 이용하려면  
NewUnstartedServer 사용

# HTTP서버 mock #3

프로메테우스  
mock 사례

httptest 활용  
http.mux 활용  
gin 코드 참고

```
// prometheus.go
```

```
func New() (*mocked.Server, error) {  
    s := mocked.New()  
    s.GET("/api/v1/metadata", handleMetadata)  
    s.GET("/api/v1/query", handleQuery)  
    s.GET("/api/v1/query_range", handleQueryRange)  
    s.GET("/api/v1/status/buildinfo", handleBuildInfo)  
    err := s.Start()  
    if err != nil {  
        err = fmt.Errorf("start err: %w", err)  
    }  
    return s, err  
}
```

```
func handleMetadata(c *mocked.Context) {  
    c.JSONString(200, `{"status":"success","data":{"...}}`)  
}
```

```
func handleQuery(c *mocked.Context) {  
    query := c.Query("query")  
    if query == "" {  
        c.JSONString(405, `{"status":"error","errorType":"bad_data",...}`)  
        return  
    }  
    if query == "up" {  
        c.JSONString(200, `{"status":"success","data":{"resultType":"vector"}}`)  
        return  
    }  
}
```

...

<https://github.com/kuoss/venti/blob/v0.2.6/pkg/mocked/prometheus/prometheus.go>



# 14. TestMain & suite

수명주기: 셋업(setup) → 테스트 → 해체(teardown)

- TestMain - 간결하고 표준적인 셋업/해체 ★
- testify/suite - 약간 복잡하지만 다양한 수준별 셋업/해체 기능

※ 간단한 셋업은 init()으로도 가능(해체는 불가)



# TestMain & suite

TestMain - 표준 라이브러리 testing 사용, 함수 형식  
suite - 서드파티 testify 하위 패키지, 메소드 형식

```
func TestMain(m *testing.M) { // 진입점
    setup()
    code := m.Run()
    teardown()
    os.Exit(code)
}

func setup() {
    // 셋업 코드
}

func teardown() {
    // 해체 코드
}

func TestXxx(t *testing.T) {
    // 테스트 함수
}
```

```
type ExampleTestSuite struct {
    suite.Suite
    // 멤버 변수 추가 가능
}

func (suite *ExampleTestSuite) SetupSuite() {
    // 셋업 코드
}

func (suite *ExampleTestSuite) TearDownSuite() {
    // 해체 코드
}

func (suite *ExampleTestSuite) TestXxx() {
    // 테스트 함수
}

func TestExampleTestSuite(t *testing.T) { // 진입점
    suite.Run(t, new(ExampleTestSuite))
}

// 이외에도 Test별: SetupTest/TearDownTest
// SubTest별: SetupSubTest/TearDownSubTest ...
```



# TestMain 예시

```
// mytestmain.go
var host = "http://prometheus.example.com:9090"

func GetMetadata() (string, error) {
    resp, err := http.Get(host + "/api/v1/metadata")
    if err != nil {
        return "", fmt.Errorf("http.Get err: %w", err)
    }
    defer resp.Body.Close()
    bodyBytes, err := io.ReadAll(resp.Body)
    if err != nil {
        return "", fmt.Errorf("io.ReadAll err: %w", err)
    }
    return string(bodyBytes), nil
}

// 프로메테우스 메타데이터 조회하는 함수
```

```
// mytestmain_test.go
var server *mockers.Server
```

```
func TestMain(m *testing.M) { 진입점
    setup()
    code := m.Run()
    teardown()
    os.Exit(code)
}
```

```
func setup() { [셋업]
    var err error 프로메테우스 mock 기동
    server, err = prometheus.New()
    if err != nil {
        panic(err)
    }
    host = server.URL
}
```

```
func teardown() { [해체]
    server.Close() 프로메테우스 mock 종료
}
```

```
func TestGetMetadata(t *testing.T) {
    want := `{"status": "success", "data": ...}`
    got, err := GetMetadata()
    assert.NoError(t, err)
    assert.Equal(t, want, got)
}
```

# suite

앞의 TestMain

예시를 suite로 구현

```
type ExampleTestSuite struct {
    suite.Suite
    server *mockers.Server  멤버 변수 추가
}

func (suite *ExampleTestSuite) SetupSuite() { 셋업
    var err error
    suite.server, err = prometheus.New()
    if err != nil {
        panic(err)
    }
    host = suite.server.URL
}

func (suite *ExampleTestSuite) TearDownSuite() { 해체
    suite.server.Close()
}

func (suite *ExampleTestSuite) TestGetMetadata() { 테스트 함수
    want := `{"status":"success","data":...}`
    got, err := GetMetadata()
    suite.NoError(err)
    suite.Equal(want, got)
}

func TestExampleTestSuite(t *testing.T) { 진입점
    suite.Run(t, new(ExampleTestSuite))
}
```

# TestMain 사례

※ 해체, 꼭 필요한가?  
그렇진 않지만,  
mock 서버 여러 대 구동  
× 여러 테스트 병렬 실행시  
리소스 부족, 성능 저하

3

venti의 remoteHandler 테스트  
[https://github.com/kuoss/venti/blob/v0.2.6/pkg/handler/remote/remote\\_test.go](https://github.com/kuoss/venti/blob/v0.2.6/pkg/handler/remote/remote_test.go)

GopherCon Korea 2023

```
func TestMain(m *testing.M) {  
    ...  
}  
  
func shutdown() {    [해체]  
    servers.Close()  mock서버 종료  
}  
  
func setup() {    [셋업]  
    servers = ms.New(ms.Requirements{    테스트에 필요한 mock서버 기동  
        {Type: ms.TypePrometheus, Name: "prometheus1", IsMain: true},    ( prometheus 2대, lethe 2대 )  
        {Type: ms.TypePrometheus, Name: "prometheus2", IsMain: false},  
        {Type: ms.TypeLethe, Name: "lethe1", IsMain: true},  
        {Type: ms.TypeLethe, Name: "lethe2", IsMain: false},  
    })  
    ...  
    remoteHandler1 = New(datasourceService, remoteService)  
  
    remoteRouter = gin.New()  
    api := remoteRouter.Group("/api")    router 생성하고 handler 연결  
    api.GET("/remote/metadata", remoteHandler1.Metadata)  
    api.GET("/remote/query", remoteHandler1.Query)  
    api.GET("/remote/query_range", remoteHandler1.QueryRange)  
}  
...
```



# V. IDE & 커버리지

- 15. vscode 활용
- 16. 커버리지
- 17. 커버리지 100%?



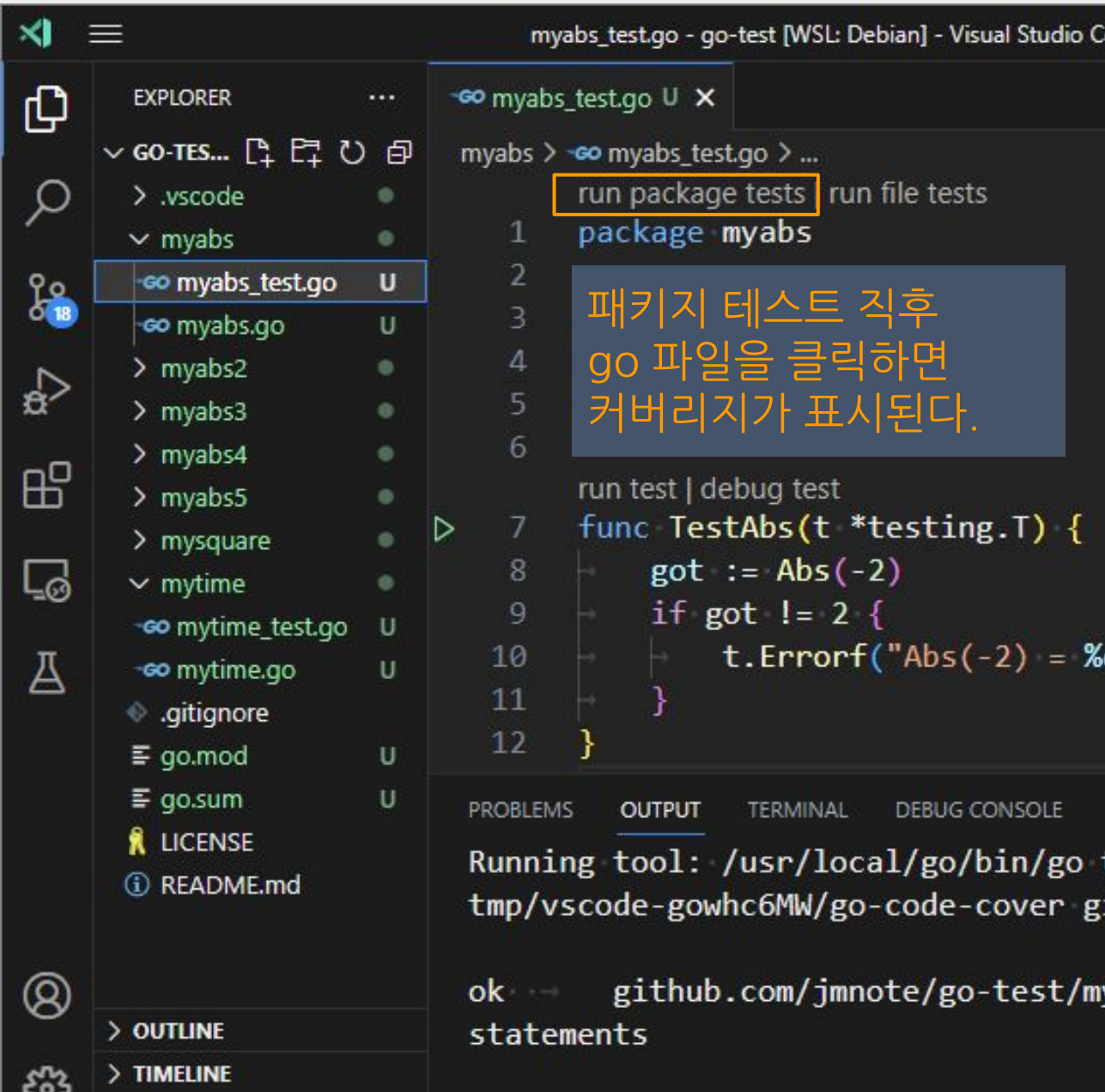
# 15. vscode 활용

IDE를 잘 쓰면 테스트가 편해진다.





# vscode #1 패키지 테스트 실행



myabs\_test.go - go-test [WSL: Debian] - Visual Studio Code

EXPLORER

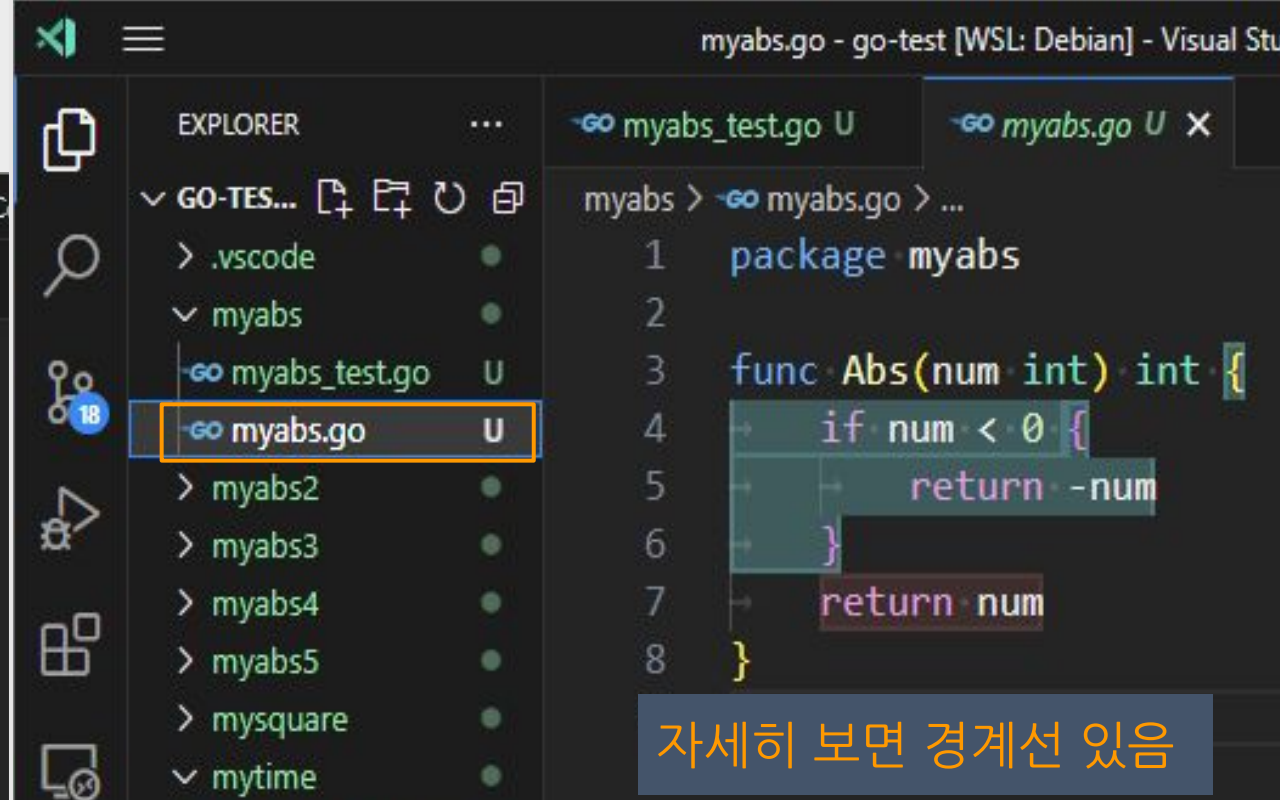
- GO-TEST...
- .vscode
- myabs
  - myabs\_test.go U
  - myabs.go U
  - myabs2
  - myabs3
  - myabs4
  - myabs5
  - mysquare
  - mytime
  - mytime\_test.go U
  - mytime.go U
- .gitignore
- go.mod U
- go.sum U
- LICENSE
- README.md

run package tests run file tests

package myabs

패키지 테스트 직후  
go 파일을 클릭하면  
커버리지가 표시된다.

```
1 package myabs
2
3
4
5
6
7 run test | debug test
8 func TestAbs(t *testing.T) {
9     got := Abs(-2)
10    if got != 2 {
11        t.Errorf("Abs(-2) = %d; want 2", got)
12    }
13 }
```



myabs.go - go-test [WSL: Debian] - Visual Studio Code

EXPLORER

- GO-TEST...
- .vscode
- myabs
  - myabs\_test.go U
  - myabs.go U
  - myabs2
  - myabs3
  - myabs4
  - myabs5
  - mysquare
  - mytime

package myabs

```
1 package myabs
2
3 func Abs(num int) int {
4     if num < 0 {
5         return -num
6     }
7     return num
8 }
```

자세히 보면 경계선 있음

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Go Tests

Running tool: /usr/local/go/bin/go test -timeout 30s -coverprofile=/tmp/vscode-gowhc6MW/go-code-cover github.com/jmnote/go-test/myabs

ok -- github.com/jmnote/go-test/myabs 0.002s coverage: 66.7% of statements

## vscode #2 커버리지

```
PROBLEMS OUTPUT TEST RESULTS TERMINAL ... bash - go-test

● root@ws1:~/go/src/go-test# /usr/local/go/bin/go test -timeout 30s -coverprofile=/tmp/vs
code-gowhc6MW/go-code-cover github.com/jmnote/go-test/myabs
ok      github.com/jmnote/go-test/myabs 0.001s coverage: 66.7% of statements
● root@ws1:~/go/src/go-test# cat /tmp/vscode-gowhc6MW/go-code-cover
mode: set
github.com/jmnote/go-test/myabs/myabs.go:3:23,4:13 1 1
github.com/jmnote/go-test/myabs/myabs.go:4:13,6:3 1 1
github.com/jmnote/go-test/myabs/myabs.go:7:2,7:12 1 0
○ root@ws1:~/go/src/go-test#
○ root@ws1:~/go/src/go-test#
○ root@ws1:~/go/src/go-test#
```

```
1 package myabs
2
3 func Abs(num int) int {
4     if num < 0 {
5         return -num
6     }
7     return num
8 }
```

구간 3개

3행23열~4행13열 1회 실행 (녹색)  
4행13열~6행 3열 1회 실행 (녹색)  
7행 2열~7행12열 0회 실행 (빨간색)

열은 바이트 단위로서, 에디터와 약간 차이가 있다. (탭은 4칸이지만, 1바이트)

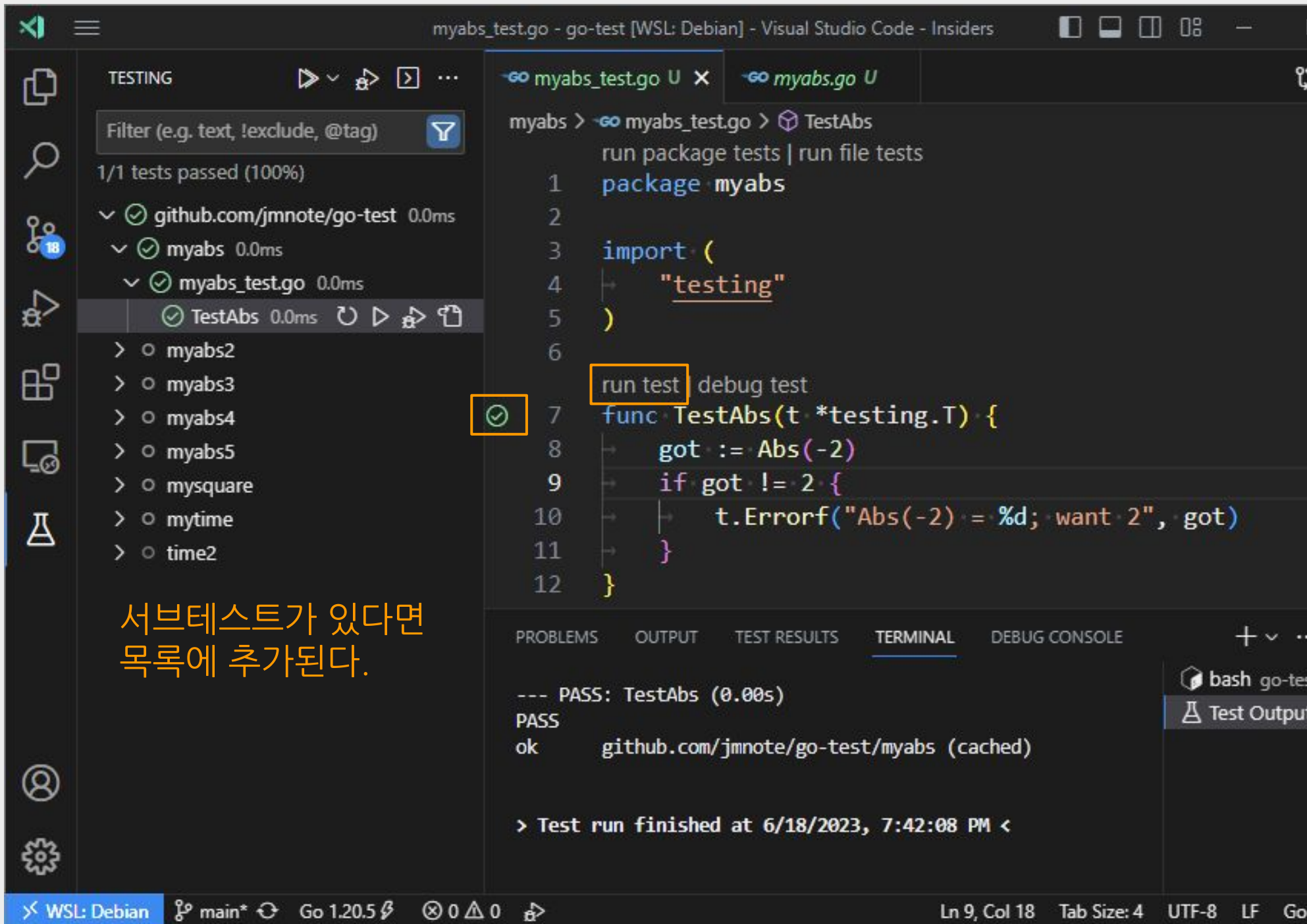


# vscode #3

## 테스트함수 실행

테스트함수의  
위쪽 run test 또는  
왼쪽 ▷ (또는 ⑤)  
클릭

▷를 클릭하면  
왼쪽에 TESTING  
패널이 열린다.





# vscode #4

서브테스트 실행  
(테스트케이스)

TESTING 패널에서  
테스트케이스의 ▶  
클릭

The screenshot shows the Visual Studio Code interface with the TESTING panel on the left and the source code editor on the right. The TESTING panel displays a list of tests, with the test case `#2_Abs(100)` highlighted and its execution button (▶) clicked. The source code editor shows the `TestAbs_2` function, which defines test cases for the `Abs` function. The test case `{100, 99}` is highlighted in the code. The bottom panel shows the TEST RESULTS, indicating that the test run passed.

myabs8\_test.go - go-test [WSL: Debian] - Visual Studio Code - Insiders

TESTING

Filter (e.g. text, !exclude, @tag)

0/0 tests passed (0.00%)

- myabs4
- myabs5
- myabs6
- myabs7 0.0ms
- myabs8 0.0ms
  - myabs8\_test.go 0.0ms
    - TestAbs\_1 0.0ms
      - #0\_Abs(-1)
      - #1\_Abs(-2)
      - #2\_Abs(-3)
    - TestAbs\_2 0.0ms
      - #0\_Abs(0) 0.0ms
      - #1\_Abs(1) 0.0ms
      - #2\_Abs(100) 0.0ms
- myappendfloat
- mybench
- mybench2
- myclock
- myerror
- myexample
- myfuzz

myabs8 > go myabs8\_test.go > ...

```
26 |
27 | run test | debug test
28 | func TestAbs_2(t *testing.T) {
29 |     testCases := []struct {
30 |         num int
31 |         want int
32 |     }{
33 |         {0, 0},
34 |         {1, 1},
35 |         {100, 99},
36 |     }
37 |     for i, tc := range testCases {
38 |         t.Run(fmt.Sprintf("#%d Abs(%d)", i, tc.num), func(t *testing.T) {
39 |             got := Abs(tc.num)
40 |             assert.Equal(t, got, tc.want)
41 |         })
42 |     }
43 | }
```

PROBLEMS 1 OUTPUT TEST RESULTS TERMINAL DEBUG CONSOLE

ub.com/jmnote/go-test/myabs8

```
=== RUN TestAbs_2
--- PASS: TestAbs_2 (0.00s)
testing: warning: no tests to run
PASS
ok      github.com/jmnote/go-test/myabs8 (cached) [no tests to run]
```

Test run at 7/5/2023, 10:15:01 PM

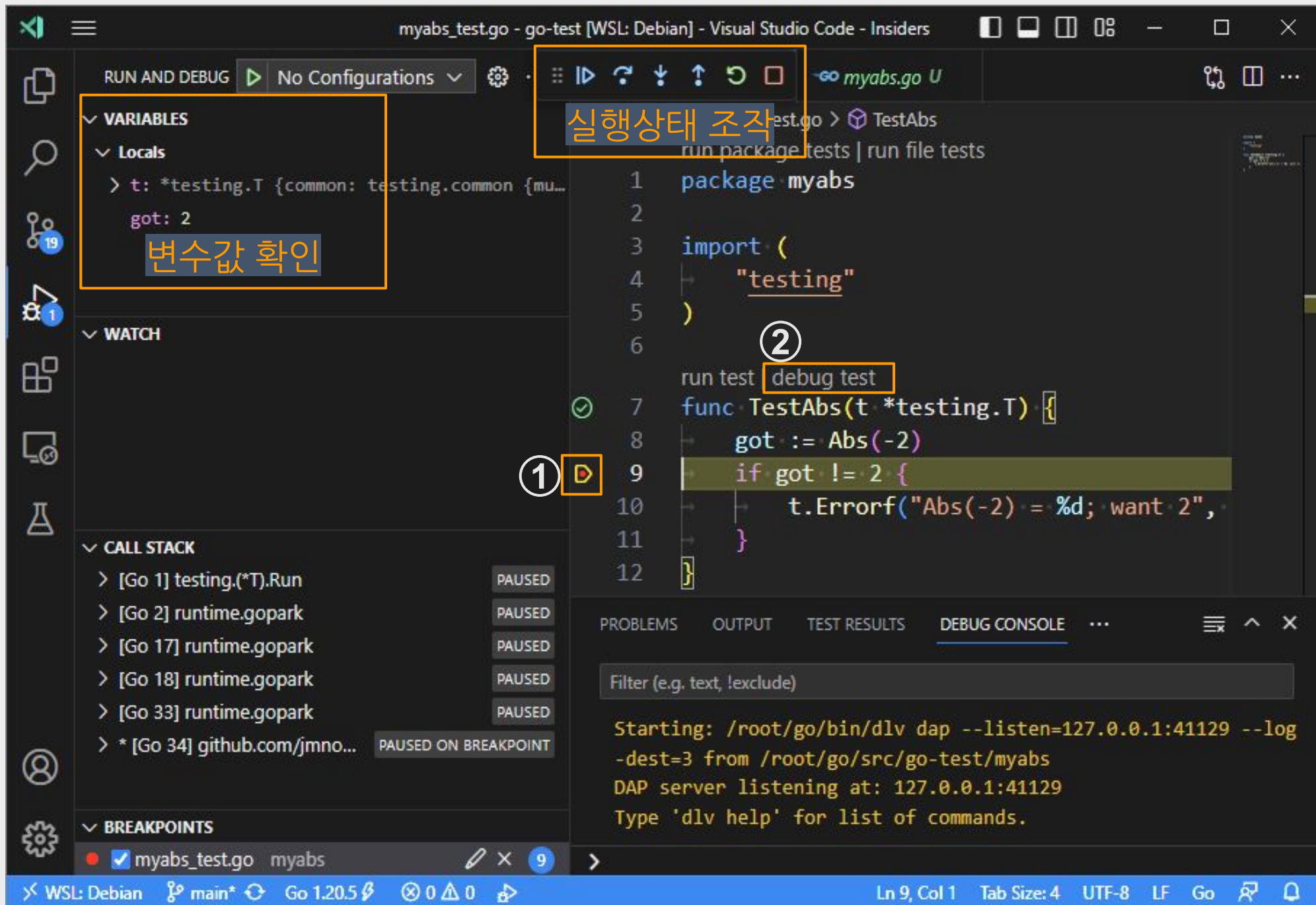
- Test run at 7/5/2023, 10:14:34 PM
- Test run at 7/5/2023, 10:13:35 PM
- Test run at 7/5/2023, 10:13:27 PM
- Test run at 7/5/2023, 10:13:21 PM
- Test run at 7/5/2023, 10:13:18 PM

WSL: Debian main\* Go 1.20.5 0 1 Ln 26, Col 1 Tab Size: 4 UTF-8 LF Go

# vscode #5

## 디버그 테스트

- ① 중단점 설정
- ② debug test 클릭





# 16. 커버리지

커버리지 원리, 모드, 도구에 대해 알아보자.

※ 참고자료: <https://go.dev/blog/cover>



# 커버리지 예시

```
// mycover.go
func Size(a int) string {
    switch {
    case a < 0:
        return "negative"
    case a == 0:
        return "zero"
    case a < 10:
        return "small"
    case a < 100:
        return "big"
    case a < 1000:
        return "huge"
    }
    return "enormous"
}
```

```
// mycover_test.go
func TestSize(t *testing.T) {
    testCases := []struct {
        in    int
        want  string
    }{
        {-1, "negative"},
        {5, "small"},
    }
    for _, tc := range testCases {
        got := Size(tc.in)
        assert.Equal(t, tc.want, got)
    }
}
```

```
root@wsl:~/go/src/go-test/mycover# go test -cover
PASS
```

```
ok      github.com/jmnote/go-test/mycover
        github.com/jmnote/go-test/mycover
```

```
coverage: 42.9% of statements
0.002s
```

# 커버리지 측정 원리

```
// mycover.go
func Size(a int) string {
    switch {
    case a < 0:
        return "negative"
    case a == 0:
        return "zero"
    case a < 10:
        return "small"
    case a < 100:
        return "big"
    case a < 1000:
        return "huge"
    }
    return "enormous"
}
```

재작성

```
// mycover.go (rewritten)
func Size(a int) string {
    GoCover.Count[0] = 1
    switch {
    case a < 0:
        GoCover.Count[2] = 1
        return "negative"
    case a == 0:
        GoCover.Count[3] = 1
        return "zero"
    case a < 10:
        GoCover.Count[4] = 1
        return "small"
    case a < 100:
        GoCover.Count[5] = 1
        return "big"
    case a < 1000:
        GoCover.Count[6] = 1
        return "huge"
    }
    GoCover.Count[1] = 1
    return "enormous"
}
```

# go test -cover

# go test -coverprofile

→ 커버리지 활성화되면  
go test는 소스코드 재작성  
(실행횟수 Count 문 추가)

테스트가 완료되면 카운트를  
수합되어 백분율을 계산한다.

※ 이 방식이 오버헤드가 클 것  
같지만, 약 3% 정도라고 한다.



# 커버리지 모드 3가지

모드	설명	Count 값
set	각 문장이 실행되었는가? (기본 모드)	0 또는 1
count	각 문장이 몇번 실행되었는가?	0 ~ N
atomic	count와 같지만 병렬 프로그램에서 더 정확하게 카운트	0 ~ N

기본 모드는 set

명령어에 플래그를 붙여 다른 모드 적용 가능

```
# go test -covermode=count
```

※ 어느 모드를 선택하든 커버리지 백분율에는 영향이 없다. ★  
즉, cover를 했는지(0인지 아닌지)만 보고 계산한다.

# 커버리지 프로파일 (set 모드)

```
github.com/jmnote/go-test/mycover/mycover.go (42.9%) not tracked not covered covered

package mycover

func Size(a int) string {
    switch {
    case a < 0:
        return "negative"
    case a == 0:
        return "zero"
    case a < 10:
        return "small"
    case a < 100:
        return "big"
    case a < 1000:
        return "huge"
    }
    return "enormous"
}
```

```
# go test -coverprofile=coverage.out
PASS
github.com/jmnote/go-test/mycover coverage: 42.9% of statements
ok github.com/jmnote/go-test/mycover 0.002s
```

```
# cat coverage.out
```

```
mode: set 첫행은 커버리지 모드
github.com/jmnote/go-test/mycover/mycover.go:3.25,4.9 1 1
github.com/jmnote/go-test/mycover/mycover.go:5.13,6.20 1 1
... 파일명.go:시작행.열,끝행.열 | count
                                numberOfStatements
```

```
# go tool cover -func=coverage.out
github.com/jmnote/go-test/mycover/mycover.go:3: Size 42.9%
total: (statements) 42.9%
```

```
# go tool cover -html=coverage.out -o coverage.html
```

profile 파일 형식

<https://cs.opensource.google/go/x/tools/+refs/tags/v0.11.0:cover/profile.go;l=55-58>

# 히트맵 (count 모드)

```
fmt/format.go  not tracked  no coverage  low coverage  * * * * * high coverage

// pad appends b to f.buf, padded on left (w > 0) or right (w < 0 or f.minus).
func (f *fmt) pad(b []byte) {
    if !f.widPresent || f.wid == 0 {
        f.buf.Write(b)
        return
    }
    padding, left, right := f.computePa
    if left > 0 {
        f.writePadding(left, paddin
    }
    f.buf.Write(b)
    if right > 0 {
        f.writePadding(right, paddi
    }
}
```

※ fmt 패키지에 대한 커버리지 예시

```
# go test -covermode=count -coverprofile=count.out fmt
ok      fmt 0.056s coverage: 91.7% of statements
```

```
# go tool cover -func=count.out
fmt/format.go: init          100.0%
fmt/format.go: clearflags    100.0%
fmt/format.go: init          100.0%
...
fmt/scan.go:  advance        96.2%
fmt/scan.go:  doScanf        96.8%
total:          (statements)  91.7%
```

```
# go tool cover -html=count.out -o coverage.html
```





# 17. 커버리지 100%?

- cover하기 어려운 경우, 포기하고 주석 달기
  - monkey patch, or fakeErr
  - 끝나지 않는 함수 → 고루틴을 이용한 스모크 테스트
  - Exit 테스트, Profile 조작
- ※ 커버리지 100% 꼭 필요한가? 그렇지 않지만... ~~재미가 있다.~~



# cover하기 어려운 경우

```
// myunreachable.go
type Data struct {
    Message string `json:"message"`
}

func JSONString(data Data) (string, error) {
    bytes, err := json.Marshal(data)
    if err != nil {
        return "", err // unreachable
    }
    return string(bytes), nil
}
```

Data → JSON문자열 변환

표준 라이브러리(encoding/json)에서는  
math.NaN() 같은 값을 넣어서 테스트한다.  
위 코드에서는 타입(Data) 고정되어 적용불가

```
// myunreachable_test.go
func TestJSONString(t *testing.T) {
    testCases := []struct {
        data Data
        want string
    }{
        {Data{}, `{"message":""}`},
        {Data{Message: "foo"}, `{"message":"foo"}`},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := JSONString(tc.data)
            assert.NoError(t, err)
            assert.Equal(t, tc.want, got)
        })
    }
}
```

굳이 cover할 필요가 없다면,  
커버리지 100%를 포기하고 주석만 달아두자.  
//go:cover ignore 등 제안이 있었으나 Closed.  
<https://github.com/golang/go/issues/53271>

# monkey patch

앞에서 다룬 myclock 사례, mock 교체와 같은 기법.  
json.Marshal처럼 안정적인 것까지 교체해가며 테스트해야 할지는...

```
// myunreachable2.go
var jsonMarshal = json.Marshal 변수화

type Data struct {
    Message string `json:"message"`
}

func JSONString(data Data) (string, error) {
    bytes, err := jsonMarshal(data)
    if err != nil {
        return "", err
    }
    return string(bytes), nil
}
```

```
// myunreachable2_test.go
func TestJSONString_error(t *testing.T) {
    jsonMarshal = func(v any) ([]byte, error) {
        return []byte{}, errors.New("fake")
    } 에러 발생 함수로 교체

    testCases := []struct {
        data      Data
        wantError string
    }{
        {Data{}, "fake"},
        {Data{Message: "foo"}, "fake"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := JSONString(tc.data)
            assert.EqualError(t, err, tc.wantError)
            assert.Equal(t, "", got)
        })
    }
    jsonMarshal = json.Marshal 다른 테스트를 위해 원복
}
```

# or fakeErr

커버리지 측정시 if 조건의 '&&'나 '||'까지 분해하지 않음 → 커버리지 100%

```
// myunreachable2b.go
var fakeErr bool = false fakeErr 추가

type Data struct {
    Message string `json:"message"`
}

func JSONString(data Data) (string, error) {
    bytes, err := json.Marshal(data)
    if err != nil || fakeErr {
        return "", fmt.Errorf("marshal err: %v", err)
    }
    return string(bytes), nil
}
```

```
// myunreachable2b_test.go
func TestJSONString_error(t *testing.T) {

    fakeErr = true fakeErr 활성화

    testCases := []struct {
        data      Data
        wantError string
    }{
        {Data{}, "marshal err: <nil>"},
        {Data{Message: "foo"}, "marshal err: <nil>"},
    }
    for _, tc := range testCases {
        t.Run("", func(t *testing.T) {
            got, err := JSONString(tc.data)
            assert.EqualError(t, err, tc.wantError)
            assert.Equal(t, "", got)
        })
    }
    fakeErr = false
}
```

다른 테스트를 위해 원복

# 끝나지 않는 함수

[스모크 테스트] 전원을 켜올 때 회로에서 연기가 나는가?  
→ 일정기간 실행시켰을 때 정상 동작하는가(패닉 없는가)?

```
// myhttp.go
func setupRouter() *gin.Engine {
    r := gin.Default()
    r.GET("/ping", func(c
*gin.Context) {
        c.String(200, "pong")
    })
    return r
}

func Run() {
    r := setupRouter()
    r.Run(":8080")
}
```

끝나지 않는다...  
테스트 불가?

```
// myhttp_test.go
func TestRun(t *testing.T) {
    go Run()
    time.Sleep(100 * time.Millisecond)
}

func TestRun_2(t *testing.T) {
    go Run()
    time.Sleep(100 * time.Millisecond)

    resp, err := http.Get("http://127.0.0.1:8080/ping")
    assert.NoError(t, err)
    body, err := io.ReadAll(resp.Body)
    assert.NoError(t, err)
    assert.Equal(t, string(body), "pong")
}

스모크 테스트 + GET 테스트
```

Run()을 고루틴으로 실행  
0.1초간 정상 동작 확인  
(커버리지 확보)

# Exit 테스트 #1

```
// myunreachable3.go
func Greet(ok bool) {
    if !ok {
        log.Fatal("bye")
    }
    fmt.Println("hello")
}
```

설정에 오류가 있어 종료시킬 때,  
이런 패턴이 나타날 수 있다.

Greet(false)를 테스트하려고 보니...

- bye가 stderr로 출력된다.
- 앞에 날짜시간이 붙는다.
- 종료(Exit)된다!

log.Fatal()=log.Print()+os.Exit(1)

```
// myunreachable3_test.go
import (
    ...
    "github.com/kuoss/common/tester"
)

func TestGreet_hello(t *testing.T) {
    stdout, stderr, err := tester.CaptureChildTest(func() {
        Greet(true)
    })
    assert.Equal(t, "hello\n", stdout)
    assert.Equal(t, "", stderr)
    assert.NoError(t, err)
}

func TestGreet_bye(t *testing.T) {
    stdout, stderr, err := tester.CaptureChildTest(func() {
        Greet(false)
    })
    assert.Equal(t, "", stdout)
    // 2023/07/13 22:13:07 bye
    assert.Regexp(t, `^[0-9:/ ]+ bye\n$`, stderr)
    assert.EqualError(t, err, "exit status 1")
}
```

# Exit 테스트 #2

CaptureChildTest는 어떻게 생겼을까?

<https://github.com/kuoss/common/blob/v0.1.4/tester/tester.go#L11>

```
func CaptureChildTest(f func()) (stdout string, stderr string, err error) {  
    if os.Getenv("CHILD") == "1" {  
        f()  
        os.Exit(0)  
    }  
    cmd := exec.Command(os.Args[0], "-test.run="+getTestRun())  
    cmd.Env = append(os.Environ(), "CHILD=1")  
    var stdoutBuf bytes.Buffer  
    var stderrBuf bytes.Buffer  
    cmd.Stdout = &stdoutBuf  
    cmd.Stderr = &stderrBuf  
    err = cmd.Run()  
    return stdoutBuf.String(), stderrBuf.String(), err  
}  
  
func getTestRun() string {  
    pc, _, _, _ := runtime.Caller(2)  
    name := runtime.FuncForPC(pc).Name()  
    if dot := strings.LastIndex(name, "."); dot >= 0 {  
        name = name[dot+1:]  
    }  
    return name  
}
```

테스트할 함수 받아서

환경변수 CHILD=1을 주고  
자식 프로세스로 테스트 실행

stdout과 stderr를 버퍼에 캡처하고  
Run 결과와 함께 return

호출한 곳 추적하여 이름 return

# Profile 조작

필요시 profile 파일을 조작하여, 일부를 제외할 수 있다.  
※ 트러블슈팅 사례이며, 커버리지 향상 목적으로는 하지 말자.

```
$ go version
go version go1.20.5 linux/amd64
$ git clone -b v0.2.3 https://github.com/kuoss/lethe.git
...
$ cd lethe/
$ go test ./... -coverprofile=cover.out
...
ok      github.com/kuoss/lethe/util    0.008s  coverage: 81.6% of statements

$ go tool cover -func cover.out
cover: inconsistent NumStmt: changed from 1 to 2

$ cat cover.out | grep yaccpar | grep 706
github.com/kuoss/lethe/letheql/parser/yaccpar:706.0,705.0 2 1
github.com/kuoss/lethe/letheql/parser/yaccpar:706.0,708.0 1 1
github.com/kuoss/lethe/letheql/parser/yaccpar:708.0,706.0 2 1

$ cat cover.out | grep -v yaccpar > cover2.out
$ go tool cover -func cover2.out
...
github.com/kuoss/lethe/util/time.go:16:  GetDurationFromAge      80.8%
github.com/kuoss/lethe/util/time.go:53:  FloatStringToTime        100.0%
total:                                     (statements) 70.9%
```

(나중에 go 버전 올라가면 해결될 듯)

재현 환경 셋업

go tool cover 비정상

???

yaccpar 제외하면...  
정상 (편안)

~~성적이 안 나오면?~~  
~~성적표를 고친다.~~

원인?  
cover 모듈이 가끔  
//line 지시자 처리를  
못하는 현상  
<https://github.com/golang/go/issues/41222>

//line 지시자?  
코드 generate시  
(예: goyacc),  
원래 코드의 위치를  
나타내기 위한 주석  
[https://github.com/kuoss/lethe/blob/v0.2.3/letheql/parser/generated\\_parser.y.go](https://github.com/kuoss/lethe/blob/v0.2.3/letheql/parser/generated_parser.y.go)





## VI. 기타

- 18. 통합 테스트
- 19. 테스트 이외의 테스트?
- 20. 참고자료



# 18. 통합 테스트

※ 지금까지 다룬 것은 대부분 단위 테스트. 여기서 다룰 것은...

- 통합 테스트 = 빌드(패키지 통합)한 파일을 가지고 하는 테스트
- Go 1.20부터 go build -cover로,  
통합 테스트에서도 커버리지 측정 가능

<https://go.dev/blog/integration-test-coverage>



# 통합 테스트 #1 수작업 테스트

```
// myintegration/main.go
func main() {
    numPtr := flag.String("num", "", "a string")
    flag.Parse()
    num, err := strconv.Atoi(*numPtr)
    if err != nil {
        fmt.Printf("error: %s is not int\n", *numPtr)
        os.Exit(1)
    }
    fmt.Printf("Abs(%d)=%d\n", num, myabs.Abs(num))
}
```

num 값 받아 Atoi 실행...

에러 나면 에러 표시

아니면 절대값 계산

```
root@wsl:~/go/src/go-test/myintegration# go build
root@wsl:~/go/src/go-test/myintegration# ./myintegration -num=-1
Abs(-1)=1
root@wsl:~/go/src/go-test/myintegration# ./myintegration -num=2
Abs(2)=2
root@wsl:~/go/src/go-test/myintegration# ./myintegration -num=foo
error: foo is not int
```

수작업 빌드  
실행(테스트)

테스트케이스 3개

# 통합 테스트 #2 스크립트로 테스트

```
// myintegration/test.sh
#!/bin/bash
BUILDARGS="$*" 빌드/테스트케이스 실행
go build $BUILDARGS -o myintegration .

NUMS=(-1 2 foo)
for NUM in "${NUMS[@]}"; do
    ./myintegration -num=$NUM > /dev/null
done
echo "finished processing ${#NUMS[@]} cases, no crashes"
```

```
root@ws1:~/go/src/go-test/myintegration# ./test.sh
finished processing 3 cases, no crashes
```

```
root@ws1:~/go/src/go-test/myintegration# ./wrap_test.sh
...
+ go tool covdata percent -i=covdatafiles -o=cov.txt
    github.com/jmnote/go-test/myabs coverage: 100.0% of statements
    github.com/jmnote/go-test/myintegration coverage: 100.0% of statements
+ go tool cover -func=cov.txt
github.com/jmnote/go-test/myabs/myabs.go:3:      Abs      100.0%
github.com/jmnote/go-test/myintegration/main.go:12:  main      100.0%
total:                                           (statements) 100.0%
```

```
// myintegration/wrap_test.sh
#!/bin/bash
PKGARGS="$*"
rm -rf covdatafiles; mkdir covdatafiles

GOCOVERDIR=covdatafiles ./test.sh -cover $PKGARGS
set -x
go tool covdata percent -i=covdatafiles -o=cov.txt
go tool cover -func=cov.txt
```

test.sh에 -cover 붙여  
실행하고 커버리지 확인

- ※ 이 스크립트에서는 crash 아니면 정상 (에러, exit status 1도 정상)
- ※ HTTP서버도 테스트 가능? 사례 못찾음. 종료 API 있으면 가능할 듯?



## 19. 테스트 이외의 테스트?

- 단위 테스트 코드 이외의 코드 점검 항목들...
- 코드 품질 관리에 도움이 된다.
- 자동화 점검을 통과한 PR만 리뷰하면 효율적이다.



# GitHub Actions

## PR 등록/변경시 점검 10종 사례

✓ init log\_data\_path #52

Summary

Jobs

✓ go-fmt

✓ goimports

✓ go-vet

✓ staticcheck

✓ golangci-lint

✓ go-test-failfast

✓ go-test-coverage

✓ go-licenses

✓ docker-build

Run

Workflow file for this run

.github/workflows/pull-request.yml at 1fa720d

```
1 name: pull-request
2 on:
3   pull_request:
4     types: [opened, reopened, sync
5 permissions:
6   contents: read
7   pull-requests: write
8
9 jobs:
10
11   go-fmt:
12     runs-on: ubuntu-latest
13     steps:
14       - uses: actions/checkout@v3
```

Workflow file

<https://github.com/kuoss/lethe/blob/v0.2.1/.github/workflows/pull-request.yml>

← → ↺

github.com/kuoss/lethe/pull/85 PR #85

Merged

init log\_data\_path #85

jmnote merged 2 commits into main from init-data-log-path 3 weeks ago

jmnote merged commit 01aa33c into main

Hide details

Revert

3 weeks ago

10 checks passed

✓ go-fmt

✓ goimports

✓ go-vet

✓ staticcheck

✓ golangci-lint

✓ go-test-failfast

✓ go-test-coverage

✓ go-licenses

✓ docker-build

✓ coverage/coveralls First build on init-data-log-...

go fmt 형식에 맞는가?

goimports 형식에 맞는가?

go vet에 걸리는 것이 없는가?

staticcheck에 걸리는 것이 없는가?

golangci-lint에 걸리는 것이 없는가?

go-test-failfast (fail시 빨리 종료)

go test 통과? + 커버리지 등록

재배포 가능한 라이선스만 사용?

docker build 잘 되는가?



# Go Report Card

최근 release에  
대해 점검하고  
보고서 확인

goreportcard.com/report/github.com/kuoss/lethe

Go Report Card  High Scores GitHub Supporters Patreon About

## Report for github.com/kuoss/lethe

(v0.2.1) [go report](#) **A+** [Tweet](#)

**A+** Excellent! Found 5 issues across 93 files

Results	go vet	100%
go_vet 100%	go vet examines Go source code and reports suspicious constructs, such as Printf calls whose arguments do not align with the format string.	
gofmt 100%		
gocyclo 95%	No problems detected. Good job!	
ineffassign 100%		
license 100%		100%
misspell 98%	We run gofmt -s on your code, where -s is for the "simplify" command	

Last refresh: 1 day ago

[Refresh now](#)

No problems detected. Good job!



## 20. 참고자료





# 참고자료 #1

## Go 블로그 테스트 관련 글

#	날짜	제목	링크
1	2023-03-27	Code coverage for Go <b>integration tests</b>	<a href="https://go.dev/blog/integration-test-coverage">https://go.dev/blog/integration-test-coverage</a>
2	2021-06-02	<b>Fuzzing</b> is Beta Ready	<a href="https://go.dev/blog/fuzz-beta">https://go.dev/blog/fuzz-beta</a>
3	2016-10-02	Using <b>Subtests</b> and <b>Sub-benchmarks</b>	<a href="https://go.dev/blog/subtests">https://go.dev/blog/subtests</a>
4	2015-05-06	Testable <b>Examples</b> in Go	<a href="https://go.dev/blog/examples">https://go.dev/blog/examples</a>
5	2013-12-01	The <b>cover</b> story	<a href="https://go.dev/blog/cover">https://go.dev/blog/cover</a>

## 참고자료 #2

그외 참고자료

#	제목	링크
1	go help (test, testflag, testfunc)	<a href="https://pkg.go.dev/cmd/go/internal/test">https://pkg.go.dev/cmd/go/internal/test</a>
2	testing 패키지 문서	<a href="https://pkg.go.dev/testing">https://pkg.go.dev/testing</a>
3	testify 패키지 문서	<a href="https://github.com/stretchr/testify">https://github.com/stretchr/testify</a>
4	lethe/venti 개발 경험	<a href="https://github.com/kuoss/lethe">https://github.com/kuoss/lethe</a> <a href="https://github.com/kuoss/venti">https://github.com/kuoss/venti</a> Contribution 환영합니다~!



# Q & A



# Thank you

