Ginkgo

build unknown

Jump to the docs to learn more. To start rolling your Ginkgo tests *now* keep reading!

If you have a question, comment, bug report, feature request, etc. please open a GitHub issue, or visit the Ginkgo Slack channel.

TLDR

Ginkgo builds on Go's `testing` package, allowing expressive Behavior-Driven Development ("BDD") style tests. It is typically (and optionally) paired with the Gomega matcher library.

```
Describe("the strings package", func() {
  Context("strings.Contains()", func() {
    When("the string contains the substring in the middle", func() {
      It("returns `true`", func() {
        Expect(strings.Contains("Ginkgo is awesome", "is")).To(BeTrue())
      })
    })
  })
})
```

Feature List

- Ginkgo uses Go's `testing` package and can live alongside your existing `testing` tests. It's easy to bootstrap and start writing your first tests
- Ginkgo allows you to write tests in Go using expressive Behavior-Driven Development ("BDD") style:
  - Nestable `Describe`, `Context` and `When` container blocks
  - `BeforeEach` and `AfterEach` blocks for setup and teardown
  - `It` and `Specify` blocks that hold your assertions
  - `JustBeforeEach` blocks that separate creation from configuration (also known as the subject action pattern).
  - `BeforeSuite` and `AfterSuite` blocks to prep for and cleanup after a suite.
- A comprehensive test runner that lets you:

- Mark specs as pending
- Focus individual specs, and groups of specs, either programmatically or on the command line
- Run your tests in random order, and then reuse random seeds to replicate the same order.
- Break up your test suite into parallel processes for straightforward test parallelization

* `ginkgo`: a command line interface with plenty of handy command line arguments for running your tests and generating test files. Here are a few choice examples:

  - `ginkgo -nodes=N` runs your tests in `N` parallel processes and print out coherent output in realtime
  - `ginkgo -cover` runs your tests using Go's code coverage tool
  - `ginkgo convert` converts an XUnit-style `testing` package to a Ginkgo-style package
  - `ginkgo -focus="REGEXP"` and `ginkgo -skip="REGEXP"` allow you to specify a subset of tests to run via regular expression
  - `ginkgo -r` runs all tests suites under the current directory
  - `ginkgo -v` prints out identifying information for each tests just before it runs

  And much more: run `ginkgo help` for details!

  The `ginkgo` CLI is convenient, but purely optional -- Ginkgo works just fine with `go test`

* `ginkgo watch` watches packages *and their dependencies* for changes, then reruns tests. Run tests immediately as you develop!

* Built-in support for testing asynchronicity

* Built-in support for benchmarking your code. Control the number of benchmark samples as you gather runtimes and other, arbitrary, bits of numerical information about your code.

* Completions for Sublime Text: just use Package Control to install `Ginkgo Completions`.

* Completions for VSCode: just use VSCode's extension installer to install `vscode-ginkgo`.

* Straightforward support for third-party testing libraries such as Gomock and Testify. Check out the docs for details.

* A modular architecture that lets you easily:

  - Write custom reporters (for example, Ginkgo comes with a JUnit XML reporter and a TeamCity reporter).
  - Adapt an existing matcher library (or write your own!) to work with Ginkgo

Gomega: Ginkgo's Preferred Matcher Library

Ginkgo is best paired with Gomega. Learn more about Gomega here

Agouti: A Go Acceptance Testing Framework

Agouti allows you run WebDriver integration tests. Learn more about Agouti here

Getting Started

You'll need the Go command-line tools. Follow the installation instructions if you don't have it installed.

Global installation

To install the Ginkgo command line interface into the `$PATH` (actually to `$GOBIN`):

```
go get -u github.com/onsi/ginkgo/ginkgo
```

Go module "tools package":

Create (or update) a file called `tools/tools.go` with the following contents:

```
// +build tools

package tools

import (
	_ "github.com/onsi/ginkgo"
)

// This file imports packages that are used when running go generate, or used
// during the development process but not otherwise depended on by built code.
```

The Ginkgo command can then be run via `go run github.com/onsi/ginkgo/ginkgo`. This approach allows the version of Ginkgo to be maintained under source control for reproducible results, and is well suited to automated test pipelines.

Bootstrapping

```
cd path/to/package/you/want/to/test

ginkgo bootstrap # set up a new ginkgo suite
ginkgo generate  # will create a sample test file.  edit this file and add your
tests then...

go test # to run your tests

ginkgo  # also runs your tests
```

I'm new to Go: What are my testing options?

Of course, I heartily recommend Ginkgo and Gomega. Both packages are seeing heavy, daily, production use on a number of projects and boast a mature and comprehensive feature-set.

With that said, it's great to know what your options are :)

What Go gives you out of the box

Testing is a first class citizen in Go, however Go's built-in testing primitives are somewhat limited: The testing package provides basic XUnit style tests and no assertion library.

Matcher libraries for Go's XUnit style tests

A number of matcher libraries have been written to augment Go's built-in XUnit style tests. Here are two that have gained traction:

- testify
- gocheck

You can also use Ginkgo's matcher library Gomega in XUnit style tests

BDD style testing frameworks

There are a handful of BDD-style testing frameworks written for Go. Here are a few:

- Ginkgo ;)
- GoConvey
- Goblin
- Mao
- Zen

Finally, @shageman has put together a comprehensive comparison of Go testing libraries.

Go explore!

License

Ginkgo is MIT-Licensed

Contributing

See CONTRIBUTING.md