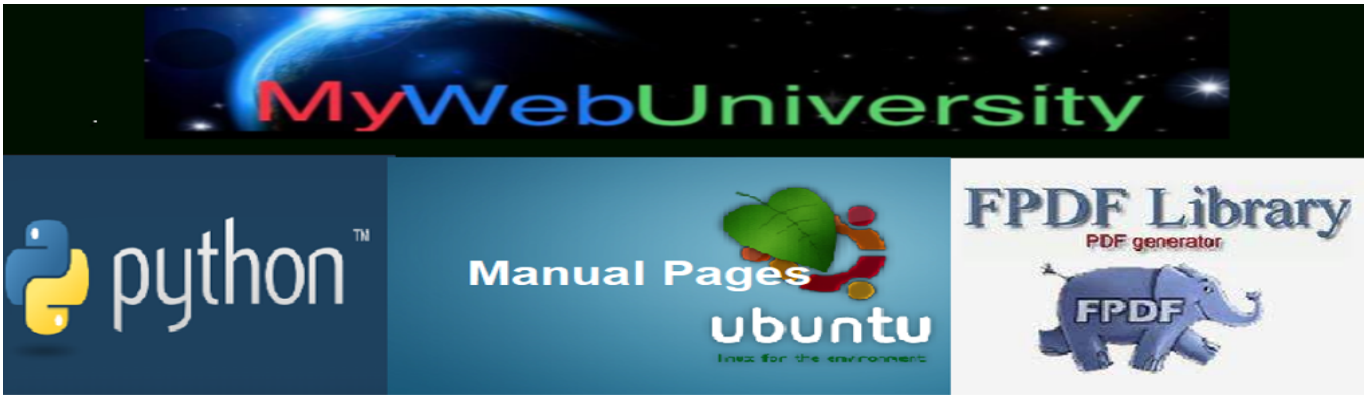

Rocky Enterprise Linux 9.2 Manual Pages on command 'go-testflag.7'

$ man go-testflag.7

GO-TESTFLAG(7) Miscellaneous Information Manual GO-TESTFLAG(7)

NAME

go - tool for managing Go source code

DESCRIPTION

The 'go test' command takes both flags that apply to 'go test' itself and flags that apply to the resulting test binary.

Several of the flags control profiling and write an execution profile suitable for "go tool pprof"; run "go tool pprof -h" for more information. The --alloc_space, --alloc_ob? jects, and --show_bytes options of pprof control how the information is presented.

The following flags are recognized by the 'go test' command and control the execution of any test:

-bench regexp

Run only those benchmarks matching a regular expression. By default, no benchmarks are run. To run all benchmarks, use '-bench .' or '-bench=.'. The regular expres? sion is split by unbracketed slash (/) characters into a sequence of regular ex? pressions, and each part of a benchmark?s identifier must match the corresponding element in the sequence, if any. Possible parents of matches are run with b.N=1 to identify sub-benchmarks. For example, given -bench=X/Y, top-level benchmarks match? ing X are run with b.N=1 to find any sub-benchmarks matching Y, which are then run in full.

-benchtime t

Run enough iterations of each benchmark to take t, specified as a time.Duration (for example, -benchtime 1h30s).

The default is 1 second (1s).

The special syntax Nx means to run the benchmark N times (for example, -benchtime 100x).

-count n

Run each test and benchmark n times (default 1).

If -cpu is set, run n times for each GOMAXPROCS value.

Examples are always run once.

-cover Enable coverage analysis.

Note that because coverage works by annotating the source code before compilation, compilation and test failures with coverage enabled may report line numbers that don't correspond to the original sources.

-covermode set,count,atomic

Set the mode for coverage analysis for the package[s] being tested. The default is "set" unless -race is enabled, in which case it is "atomic".

The values:

set: bool: does this statement run?

count: int: how many times does this statement run?

atomic: int: count, but correct in multithreaded tests;

significantly more expensive.

Sets -cover.

-coverpkg pattern1,pattern2,pattern3

Apply coverage analysis in each test to packages matching the patterns. The de? fault is for each test to analyze only the package being tested.

See 'go help packages' for a description of package patterns.

Sets -cover.

-cpu 1,2,4

Specify a list of GOMAXPROCS values for which the tests or benchmarks should be ex? ecuted. The default is the current value of GOMAXPROCS.

-failfast

Do not start new tests after the first test failure.

-list regexp

List tests, benchmarks, or examples matching the regular expression. No tests, benchmarks or examples will be run. This will only list top-level tests. No subtest

or subbenchmarks will be shown.

-parallel n

Allow parallel execution of test functions that call t.Parallel. The value of this flag is the maximum number of tests to run simultaneously; by default, it is set to the value of GOMAXPROCS. Note that -parallel only applies within a single test bi? nary. The 'go test' command may run tests for different packages in parallel as well, according to the setting of the -p flag (see 'go help build').

-run regexp

Run only those tests and examples matching the regular expression. For tests, the regular expression is split by unbracketed slash (/) characters into a sequence of regular expressions, and each part of a test's identifier must match the corre? sponding element in the sequence, if any. Note that possible parents of matches are run too, so that -run=X/Y matches and runs and reports the result of all tests matching X, even those without sub-tests matching Y, because it must run them to look for those sub-tests.

-short Tell long-running tests to shorten their run time. It is off by default but set during all.bash so that installing the Go tree can run a sanity check but not spend time running exhaustive tests.

-shuffle off,on,N

Randomize the execution order of tests and benchmarks. It is off by default. If -shuffle is set to on, then it will seed the randomizer using the system clock. If -shuffle is set to an integer N, then N will be used as the seed value. In both cases, the seed will be reported for reproducibility.

-timeout d

If a test binary runs longer than duration d, panic.

If d is 0, the timeout is disabled.

The default is 10 minutes (10m).

-v Verbose output: log all tests as they are run. Also print all text from Log and Logf calls even if the test succeeds.

-vet list

Configure the invocation of "go vet" during "go test" to use the comma-separated list of vet checks.

If list is empty, "go test" runs "go vet" with a curated list of checks believed to

be always worth addressing.

If list is "off", "go test" does not run "go vet" at all.

The following flags are also recognized by 'go test' and can be used to profile the tests during execution:

-benchmem

Print memory allocation statistics for benchmarks.

-blockprofile block.out

Write a goroutine blocking profile to the specified file when all tests are com? plete.

Writes test binary as -c would.

-blockprofilerate n

Control the detail provided in goroutine blocking profiles by calling runtime.Set? BlockProfileRate with n.

See 'go doc runtime.SetBlockProfileRate'.

The profiler aims to sample, on average, one blocking event every n nanoseconds the program spends blocked. By default, if -test.blockprofile is set without this flag, all blocking events are recorded, equivalent to -test.blockprofilerate=1.

-coverprofile cover.out

Write a coverage profile to the file after all tests have passed.

Sets -cover.

-cpuprofile cpu.out

Write a CPU profile to the specified file before exiting.

Writes test binary as -c would.

-memprofile mem.out

Write an allocation profile to the file after all tests have passed.

Writes test binary as -c would.

-memprofilerate n

Enable more precise (and expensive) memory allocation profiles by setting run? time.MemProfileRate. See 'go doc runtime.MemProfileRate'. To profile all memory allocations, use -test.memprofilerate=1.

-mutexprofile mutex.out

Write a mutex contention profile to the specified file when all tests are complete.

Writes test binary as -c would.

-mutexprofilefraction n

Sample 1 in n stack traces of goroutines holding a contended mutex.

-outputdir directory

Place output files from profiling in the specified directory, by default the direc?

tory in which "go test" is running.

-trace trace.out

Write an execution trace to the specified file before exiting.

Each of these flags is also recognized with an optional 'test.' prefix, as in -test.v.

When invoking the generated test binary (the result of ?go test -c?) directly, however,

the prefix is mandatory.

The 'go test' command rewrites or removes recognized flags, as appropriate, both before

and after the optional package list, before invoking the test binary.

For instance, the command

go test -v -myflag testdata -cpuprofile=prof.out -x

will compile the test binary and then run it as

pkg.test -test.v -myflag testdata -test.cpuprofile=prof.out

(The -x flag is removed because it applies only to the go command's execution, not to the

test itself.)

The test flags that generate profiles (other than for coverage) also leave the test binary

in pkg.test for use when analyzing the profiles.

When 'go test' runs a test binary, it does so from within the corresponding package's

source code directory. Depending on the test, it may be necessary to do the same when in?

voking a generated test binary directly.

The command-line package list, if present, must appear before any flag not known to the go

test command. Continuing the example above, the package list would have to appear before

-myflag, but could appear on either side of -v.

When 'go test' runs in package list mode, 'go test' caches successful package test results

to avoid unnecessary repeated running of tests. To disable test caching, use any test flag

or argument other than the cacheable flags. The idiomatic way to disable test caching ex?

plicitly is to use -count=1.

To keep an argument for a test binary from being interpreted as a known flag or a package

name, use -args (see 'go help test') which passes the remainder of the command line

through to the test binary uninterpreted and unaltered.

For instance, the command

go test -v -args -x -v

will compile the test binary and then run it as

pkg.test -test.v -x -v

Similarly,

go test -args math

will compile the test binary and then run it as

pkg.test math

In the first example, the -x and the second -v are passed through to the test binary un? changed and with no effect on the go command itself. In the second example, the argument math is passed through to the test binary, instead of being interpreted as the package list.

AUTHOR

This manual page was written by Michael Stapelberg <stapelberg@debian.org> and is main? tained by the Debian Go Compiler Team <team+go-compiler@tracker.debian.org> based on the output of 'go help testflag' for the Debian project (and may be used by others).