NAME

go−test − test packages

SYNOPSIS

go test [*build/test flags*] [*packages*] [*build/test flags & test binary flags*]

DESCRIPTION

'Go test' automates testing the packages named by the import paths. It prints a summary of the test results in the format:

```
ok   archive/tar   0.011s
FAIL archive/zip   0.022s
ok   compress/gzip 0.033s
...
```

followed by detailed output for each failed package.

'Go test' recompiles each package along with any files with names matching the file pattern "*_test.go". These additional files can contain test functions, benchmark functions, fuzz tests and example functions. See 'go help testfunc' for more. Each listed package causes the execution of a separate test binary. Files whose names begin with "_" (including "_test.go") or "." are ignored.

Test files that declare a package with the suffix "_test" will be compiled as a separate package, and then linked and run with the main test binary.

The go tool will ignore a directory named "testdata", making it available to hold ancillary data needed by the tests.

As part of building a test binary, go test runs go vet on the package and its test source files to identify significant problems. If go vet finds any problems, go test reports those and does not run the test binary. Only a high−confidence subset of the default go vet checks are used. That subset is: atomic, bool, buildtags, directive, errorsas, ifaceassert, nilfunc, printf, stringintconv, and tests. You can see the documentation for these and other vet tests via "go doc cmd/vet". To disable the running of go vet, use the **−vet=***off* flag. To run all checks, use the **−vet=***all* flag.

All test output and summary lines are printed to the go command's standard output, even if the test printed them to its own standard error. (The go command's standard error is reserved for printing errors building the tests.)

The go command places *$GOROOT/bin* at the beginning of $PATH in the test's environment, so that tests that execute 'go' commands use the same 'go' as the parent 'go test' command.

Go test runs in two different modes:

The first, called local directory mode, occurs when go test is invoked with no package arguments (for example, 'go test' or 'go test **−v**'). In this mode, go test compiles the package sources and tests found in the current directory and then runs the resulting test binary. In this mode, caching (discussed below) is disabled. After the package test finishes, go test prints a summary line showing the test status ('ok' or 'FAIL'), package name, and elapsed time.

The second, called package list mode, occurs when go test is invoked with explicit package arguments (for example 'go test math', 'go test ./...', and even 'go test .'). In this mode, go test compiles and tests each of the packages listed on the command line. If a package test passes, go test prints only the final 'ok' summary line. If a package test fails, go test prints the full test output. If invoked with the **−bench** or **−v** flag, go test prints the full output even for passing package tests, in order to display the requested benchmark results or verbose logging. After the package tests for all of the listed packages finish, and their output is printed, go test prints a final 'FAIL' status if any package test has failed.

In package list mode only, go test caches successful package test results to avoid unnecessary repeated running of tests. When the result of a test can be recovered from the cache, go test will redisplay the previous output instead of running the test binary again. When this happens, go test prints '(cached)' in place of the elapsed time in the summary line.

The rule for a match in the cache is that the run involves the same test binary and the flags on the command line come entirely from a restricted set of 'cacheable' test flags, defined as **−benchtime**, **−coverprofile**, **−cpu**, **−failfast**, **−fullpath**, **−list**, **−outputdir**, **−parallel**, **−run**, **−short**, **−skip**, **−timeout** and **−v**. If a run of go test has any test or non−test flags outside this set, the result is not cached. To disable test caching, use any test flag or argument other than the cacheable flags. The idiomatic way to disable test caching explicitly is to use **−count=***1*. Tests that open files within the package's module or that consult environment variables only match future runs in which the files and environment variables are unchanged. A cached test result is treated as executing in no time at all, so a successful package test result will be cached and reused regardless of **−timeout** setting.

OPTIONS

In addition to the build flags, the flags handled by 'go test' itself are:

−args Pass the remainder of the command line (everything after **−args**) to the test binary, uninterpreted and unchanged. Because this flag consumes the remainder of the command line, the package list (if present) must appear before this flag.

−c Compile the test binary to pkg.test in the current directory but do not run it (where pkg is the last element of the package's import path). The file name or target directory can be changed with the **−o** flag.

−exec *xprog*
Run the test binary using *xprog*. The behavior is the same as in 'go run'. See 'go help run' for details.

−json Convert test output to JSON suitable for automated processing. See 'go doc test2json' for the encoding details. Also emits build output in JSON. See 'go help buildjson'.

−o *file* Save a copy of the test binary to the named file. The test still runs (unless **−c** is specified). If file ends in a slash or names an existing directory, the test is written to pkg.test in that directory.

The test binary also accepts flags that control execution of the test; these flags are also accessible by 'go test'. See 'go help testflag' for details.

For more about build flags, see 'go help build'. For more about specifying packages, see 'go help packages'.

SEE ALSO

go build(1), **go vet**(1).

AUTHOR

This manual page was created using help2man and afterwards updating the output. It is maintained by the Debian Go Compiler Team <team+go−compiler@tracker.debian.org> for the Debian project (and may be used by others).