Conversion Recipe: testthat to tinytest

R. Glenn Thomas

2026-05-02

Table of contents

Conversion Recipe: testthat to tinytest

2026-05-02 09:05 PDT

This document is the reference companion to the post 'From testthat to tinytest'. The body of the post argues the trade-offs; this document is the mechanical reference. Each section is a paired before-and-after example of a single `testthat` idiom and its `tinytest` equivalent.

The recipe assumes a package whose tests live in `tests/testthat/` with the standard `tests/testthat.R` bootstrap. The target layout is `inst/tinytest/` with a `tests/tinytest.R` bootstrap. See the post body for the rationale.

1. Layout

Element	testthat	tinytest
Test directory	`tests/testthat/`	`inst/tinytest/`
File-name prefix	`test-foo.R`	`test_foo.R`
Helper-file prefix	`helper-foo.R`	`helper_foo.R`
Bootstrap	`tests/testthat.R`	`tests/tinytest.R`
DESCRIPTION	`Suggests: testthat`	`Suggests: tinytest`
edition declaration	`Config/testthat/edition: 3`	(none)

2. Bootstrap files

Before: `tests/testthat.R`

```
library(testthat)
library(mypkg)
test_check('mypkg')
```

After: `tests/tinytest.R`

```
if (requireNamespace('tinytest', quietly = TRUE)) {
  tinytest::test_package('mypkg')
}
```

The guard makes the package buildable on systems without `tinytest` installed. Without the guard, `R CMD check` would fail on a system where `tinytest` is in `Suggests` but not installed.

3. Single test_that block

Before

```
# tests/testthat/test-arithmetic.R
test_that('addition works', {
  expect_equal(1 + 1, 2)
  expect_equal(2 + 2, 4)
  expect_true(is.numeric(1 + 1))
})
```

After

```
# inst/tinytest/test_arithmetic.R
expect_equal(1 + 1, 2)
expect_equal(2 + 2, 4)
expect_true(is.numeric(1 + 1))
```

The `test_that()` wrapper, the description string, and the braces all disappear. The three assertions are now top-level calls in the file. Test reports identify each assertion by its file name and line number.

4. Multiple test_that blocks per file

Before

```
# tests/testthat/test-arithmetic.R
test_that('addition works', {
  expect_equal(1 + 1, 2)
})

test_that('subtraction works', {
  expect_equal(2 - 1, 1)
})

test_that('multiplication works', {
  expect_equal(2 * 3, 6)
})
```

After (Option A: single file)

```
# inst/tinytest/test_arithmetic.R
expect_equal(1 + 1, 2)   # addition
expect_equal(2 - 1, 1)   # subtraction
expect_equal(2 * 3, 6)   # multiplication
```

After (Option B: split into files)

```
inst/tinytest/test_addition.R
inst/tinytest/test_subtraction.R
inst/tinytest/test_multiplication.R
```

Option B is more idiomatic when the three groups exercise different source-code concerns. `tinytest` provides no mechanism for grouping inside a file; cohesion is enforced by file boundaries.

5. Shared fixtures

Before

```
# tests/testthat/helper-data.R
make_test_data <- function() {
  data.frame(x = 1:5, y = letters[1:5])
}
```

```
# tests/testthat/test-summary.R
test_that('nrow is correct', {
  df <- make_test_data()
  expect_equal(nrow(df), 5)
})
```

`testthat` automatically sources files matching `helper-*.R` before each test file.

After

```
# inst/tinytest/helper_data.R
make_test_data <- function() {
  data.frame(x = 1:5, y = letters[1:5])
}
```

```
# inst/tinytest/test_summary.R
source('helper_data.R', local = TRUE)
df <- make_test_data()
expect_equal(nrow(df), 5)
```

The `local = TRUE` argument keeps the fixture's symbols out of the global environment. `tinytest` does not source helper files automatically; the explicit `source()` is the substitute.

6. skip_on_cran and skip_if_not_installed

Before

```
test_that('integration test', {
  skip_on_cran()
  skip_if_not_installed('database_driver')
  result <- run_integration_test()
  expect_true(result$ok)
})
```

After

```
# inst/tinytest/test_integration.R
if (!nzchar(Sys.getenv('NOT_CRAN'))) {
  exit_file('Skipping on CRAN')
}
if (!requireNamespace('database_driver', quietly = TRUE)) {
  exit_file('database_driver not installed')
}
result <- run_integration_test()
expect_true(result$ok)
```

`exit_file()` halts the current file with the supplied message recorded as a skip. The check is at file scope rather than inside an assertion; this is a deliberate design choice.

7. expect_error with class matching

Before

```
test_that('errors are typed', {
  expect_error(my_function(bad_input),
               class = 'my_pkg_validation_error')
})
```

After

```
e <- expect_error(my_function(bad_input))
expect_true(inherits(e, 'my_pkg_validation_error'))
```

`tinytest::expect_error` returns the error object invisibly, which can be captured for further inspection. There is no `class` argument; the manual `inherits()` check is the substitute.

8. expect_warning and expect_message with regex

Before

```
test_that('emits a deprecation warning', {
  expect_warning(my_old_function(),
                 'deprecated.*use my_new_function')
})
```

After

```
expect_warning(
  my_old_function(),
  pattern = 'deprecated.*use my_new_function'
)
```

`tinytest`'s `expect_warning()` accepts a regex via the explicit `pattern` argument. `expect_message()` works the same way. The behaviour is similar to `testthat`, but the argument must be named.

9. expect_snapshot (no direct port)

Before

```
test_that('table output is stable', {
  expect_snapshot(print(my_summary(mtcars)))
})
```

After (manual reference comparison)

```
# inst/tinytest/test_summary.R
out <- capture.output(print(my_summary(mtcars)))
ref <- readLines(system.file('reference_output',
                             'summary_mtcars.txt',
                             package = 'mypkg'))
expect_equal(out, ref)
```

Generate the reference once with `writeLines(out, 'mypkg/inst/reference_output/summary_mtcars.txt')`. On subsequent runs the test compares against the stored file. This is the most ergonomic substitute for text snapshots, but it lacks `testthat`'s automatic regeneration on review and the diffing affordances of `waldo`.

For binary or large outputs, store an MD5 hash:

```
out <- format_some_object(x)
expect_equal(tools::md5sum(out), 'expected_hash')
```

For images, use `tinytest::expect_equal_to_reference()`:

```
expect_equal_to_reference(my_plot_object(), 'plot.rds')
```

10. expect_silent

Before

```
test_that('no message on success', {
  expect_silent(my_quiet_function())
})
```

After

```
expect_silent(my_quiet_function())
```

Identical signature, identical semantics.

11. expect_output

Before

```
test_that('prints expected line', {
  expect_output(my_function(), 'Result: 42')
})
```

After

```
expect_stdout(my_function(), pattern = 'Result: 42')
```

`tinytest` separates `expect_stdout` and `expect_message` where `testthat` uses `expect_output` for both stdout and condition messages. Migration requires deciding which output stream the test was actually checking.

12. Assertions in testthat but not in tinytest

A class of patterns the original draft of this recipe missed. Two real migrations (zzpower, 10 files, 602 assertions; and zztable1, 13 files, 564 assertions) turned up nine `testthat` functions whose names tinytest does not export: `expect_s3_class`, `expect_type`, `expect_named`, `expect_no_error`, `expect_lt`, `expect_gt`, `expect_lte`, `expect_gte`, and `expect_is`. Each requires manual substitution. The substitutes are mechanical but worth naming explicitly because forgetting them produces a deceptively-passing migration: when `pkgload::load_all()` auto-attaches the testthat namespace from `Suggests`, the testthat versions silently resolve, and the tinytest run reports no errors but undercounts assertions.

expect_s3_class

```
# testthat
expect_s3_class(x, 'data.frame')
expect_s3_class(ui_obj, 'shiny.tag.list')
```

```
# tinytest
expect_inherits(x, 'data.frame')
expect_inherits(ui_obj, 'shiny.tag.list')
```

`expect_inherits()` is a tinytest-native assertion with the same semantics as `expect_s3_class()`.

expect_type

```
# testthat
expect_type(server_func, 'closure')
expect_type(x, 'double')
expect_type(y, 'character')
```

```
# tinytest
expect_true(is.function(server_func))
expect_equal(typeof(x), 'double')
expect_equal(typeof(y), 'character')
```

For the 'closure' case, `is.function()` is more semantic. For other type checks, comparing `typeof()` to a literal string preserves the testthat assertion verbatim.

expect_named

```
# testthat
expect_named(results_df, c('effect_size', 'cohens_d', 'power'))
```

```
# tinytest
expect_equal(sort(names(results_df)),
             sort(c('effect_size', 'cohens_d', 'power')))
```

The `sort()` is necessary because `expect_named()` defaults to order-insensitive matching. If the testthat call passed `ignore.order = FALSE`, drop the `sort()` calls on both sides.

expect_no_error

```r
# testthat
expect_no_error({
  result <- some_function(...)
  expect_true(is.character(result))
  expect_true(nchar(result) > 0)
})
```

```r
# tinytest
result <- some_function(...)
expect_true(is.character(result))
expect_true(nchar(result) > 0)
```

`expect_no_error()` is a wrapper that asserts the block neither errors nor returns a particular value. Since any uncaught error in a tinytest file fails the file by default, the wrapper is redundant: simply unwrap the block. The inner `expect_*` calls remain as direct assertions. Note that removing the outer `expect_no_error()` also removes one assertion from the count for each block; this is the expected behaviour, not a regression.

expect_lt, expect_gt, expect_lte, expect_gte

```r
# testthat
expect_lt(elapsed, 0.5)
expect_gte(speedup_ratio, 0.5)
expect_lt(memory_size, 1000000, 'Memory regression')
```

```r
# tinytest
expect_true(elapsed < 0.5)
expect_true(speedup_ratio >= 0.5)
expect_true(memory_size < 1000000)
```

The four ordering assertions have no tinytest equivalent; substitute `expect_true` with the inline comparison. The optional message argument that testthat accepts as a third positional argument has no tinytest counterpart and is dropped (the failing line and value are sufficient diagnostics in the tinytest report).

expect_is

```r
# testthat (deprecated alias of expect_s3_class)
expect_is(theme, 'table1_theme')
expect_is(output, 'character')
```

```
# tinytest
expect_inherits(theme, 'table1_theme')
expect_inherits(output, 'character')
```

`expect_is` is testthat's older alias and behaves like `expect_s3_class`. The tinytest substitute is the same: `expect_inherits`. Note that `expect_inherits` works for both S3 classes and base R types because R's class system treats 'character', 'numeric', etc. as classes for `inherits()` purposes.

Argument-name differences in `expect_error` / `expect_warning` / `expect_message`

```
# testthat (both names accepted)
expect_error(f(), regex = 'pattern')
expect_warning(f(), regexp = 'pattern')
```

```
# tinytest
expect_error(f(), pattern = 'pattern')
expect_warning(f(), pattern = 'pattern')
```

testthat accepts `regex =` and `regexp =` as named arguments to its condition-matching expect functions. tinytest accepts only `pattern =`. A textual rewrite must rename both.

Detection

Before the first dry-run, grep for the patterns:

```
grep -hoE 'expect_(s3_class|type|named|no_error|lt|gt|lte|gte|is)\b' \
  inst/tinytest/*.R | sort | uniq -c
grep -hoE '(regex|regexp)[[:space:]]*=' inst/tinytest/*.R | \
  sort | uniq -c
```

If the count is non-zero, fix them before running the suite. A pkgload-based dry run will report 'all ok' even when these calls are present, because pkgload auto-attaches testthat. The cleaner verification is to run with testthat detached:

```
suppressMessages(pkgload::load_all('.', quiet = TRUE))
if ('testthat' %in% .packages()) {
  detach('package:testthat', unload = TRUE)
}
tinytest::run_test_dir('inst/tinytest', verbose = 0)
```

13. local_edition and edition_get

`testthat` 3 introduced editions to manage breaking changes. `tinytest` has no equivalent. Calls to `local_edition()`, `edition_get()`, and `Config/testthat/edition` should be removed entirely; the resulting tests run under whatever semantics `tinytest::expect_*()` provides.

14. Parameterised tests

Before (testthat with patrick)

```
patrick::with_parameters_test_that(
  'arithmetic identities hold',
  {
    expect_equal(op(1, 0), 1)
  },
  op = list(`+`, `-`, `*`, `/`)
)
```

After

```
for (op in list(`+`, `-`, `*`, `/`)) {
  expect_equal(op(1, 0), 1)
}
```

Or, with file-level granularity:

```
ops <- list(`+`, `-`, `*`, `/`)
results <- vapply(ops, function(op) op(1, 0), numeric(1))
expect_equal(results, c(1, 1, 0, Inf))
```

The for-loop variant lacks `patrick`'s test-name interpolation, so a failing iteration is identified only by line number.

15. tolerance defaults

A subtle compatibility issue: `tinytest::expect_equal()` uses a `tolerance` of `sqrt(.Machine$double.eps)`, matching `base::all.equal()`. `testthat::expect_equal()` historically used the same default but moved to stricter equality in edition 3. Tests that pass under `testthat` 3 may need an explicit `tolerance = 0` on the `tinytest` side, and vice versa.

To assert exact equality:

```
expect_identical(x, y)
```

To assert numeric equality with a custom tolerance:

```
expect_equal(x, y, tolerance = 1e-10)
```

16. DESCRIPTION

Before

```
Suggests:
    testthat (>= 3.0.0),
    knitr,
    rmarkdown
Config/testthat/edition: 3
```

After

```
Suggests:
    tinytest,
    knitr,
    rmarkdown
```

Three changes: replace `testthat (>= 3.0.0)` with `tinytest`, remove the `Config/testthat/edition: 3` line, and do not add a version pin (`tinytest`'s API is stable enough that a floor is rarely needed).

17. Continuous integration

Before (GitHub Actions, standard r-lib)

```
- uses: r-lib/actions/check-r-package@v2
```

After

The same action works without modification because both frameworks expose their suite through `R CMD check`. The one optional addition, useful for a structured per-test report:

```
- name: Run tinytest with structured output
  run: |
    Rscript -e 'res <- tinytest::test_package("mypkg")
                print(as.data.frame(res))'
```

This step writes a per-assertion data frame to the action log, which makes failures easier to diagnose than the default `R CMD check` summary.

18. Coverage reporting

Both frameworks support `covr` without configuration:

```
covr::package_coverage()
```

`covr` instruments source files, not test files, so the choice of test framework does not affect coverage measurement.

19. Running tests interactively

Action	testthat	tinytest
Run full suite	`devtools::test()`	`tinytest::test_package('.')`
Run single file	`testthat::test_file('foo.R')`	`tinytest::run_test_file('inst/tinytest/foo.R')`
Run during R CMD check	`R CMD check` (auto)	`R CMD check` (auto)
Continuous mode	`testthat::auto_test_package()`	(none; use `entr` or similar)
Test the installed package	`testthat::test_package('mypkg')`	`tinytest::test_package('mypkg')`

20. Visibility of non-exported package objects

A structural difference between the two frameworks that the mechanical conversion does not surface until the migrated suite is exercised by `R CMD check` against an installed package. This section was added after the recipe was used on the `zzpower` package and the issue showed up.

testthat's `test_check(pkg)` runs each test file with the package's internal namespace exposed. A test file can reference non-exported objects (helper functions, package constants, internal S3 methods) by their bare names, and they resolve correctly. tinytest's `test_package(pkg)` calls `library(pkg)`, which only attaches exports. The same bare reference in a tinytest file fails with 'object not found'.

A common pattern in testthat suites is to assert the existence of a package-internal constant:

```
# tests/testthat/test-constants.R
test_that('constants are defined', {
  expect_true(exists('PKG_CONSTANTS'))
  consts <- PKG_CONSTANTS
  expect_equal(consts$THRESHOLD, 0.05)
})
```

Mechanically converted to tinytest:

```
# inst/tinytest/test_constants.R
expect_true(exists('PKG_CONSTANTS'))
consts <- PKG_CONSTANTS
expect_equal(consts$THRESHOLD, 0.05)
```

If `PKG_CONSTANTS` is non-exported, the converted file passes under `pkgload::load_all() + tinytest::run_test_dir()` (because pkgload exposes the namespace) but fails under `R CMD check` (which uses `tinytest::test_package()` against the installed package, exposing only exports).

Three remediations, in increasing order of intrusiveness on the package itself:

1. **Qualify the reference in the test file.** Replace bare names with `getFromNamespace()`:

   ```
   consts <- getFromNamespace('PKG_CONSTANTS', 'pkg')
   expect_equal(consts$THRESHOLD, 0.05)
   ```

   This works without any change to the package and is the recommended fix when the object should remain internal.

2. **Use the `pkg:::name` triple-colon operator.** Slightly shorter than `getFromNamespace`:

   ```
   expect_equal(pkg:::PKG_CONSTANTS$THRESHOLD, 0.05)
   ```

   `R CMD check` warns about `:::` use in package code but not in test files.

3. **Export the object.** Add an `@export` roxygen tag to the source declaration and regenerate NAMESPACE. Use only when the object is genuinely part of the public API.

The diagnostic that distinguishes this issue from a real test bug: under `pkgload::load_all() + tinytest::run_test_dir()` the suite passes; under `R CMD check` (or `tinytest::build_install_test()`) the suite fails with 'object not found' for one or more package-internal names.

21. Migration order

A practical sequence:

1. Create a feature branch.
2. Install `tinytest`; add it to `Suggests`.
3. Create `inst/tinytest/` and `tests/tinytest.R`.
4. Convert one test file at a time, leaving the original in place. Both bootstraps run during `R CMD check`, so the suite is double-counted during the transition. Track the conversion with a checklist.
5. After the last file is converted, delete `tests/testthat/` and `tests/testthat.R`.
6. Remove `testthat` from `Suggests` and remove `Config/testthat/edition`.
7. Run `R CMD check --as-cran` and confirm no warnings.
8. Merge.

If the migration stalls (snapshot tests, IDE dependency, parallelisation), revert the branch. The original suite is preserved on `main`.

22. When not to convert

- Snapshot-heavy suites (more than five `expect_snapshot_*()` calls). The manual reference-file pattern is workable but noticeably less ergonomic.
- Suites that depend on RStudio's test pane for interactive use. `tinytest` integrates with `R CMD check` but not with the IDE test runner.
- Suites that rely on `testthat`'s parallel mode for acceptable wall-clock time. `tinytest` runs serially by default; the `cl` argument to `test_all()` provides a manual cluster, but it is less polished.
- Packages whose contributors are unfamiliar with `tinytest`. The migration imposes a small learning cost on every future contributor; for actively-developed packages with multiple authors, the cost may exceed the benefit.

Rendered on 2026-05-02 at 09:05 PDT. Source: ~/Dropbox/prj/qblog/posts/62-testthat-to-tinytest/testthat-to-tinytest/docs/testthat-to-tinytest-recipe.qmd