From testthat to tinytest: Converting an R Package Test Suite

R. Glenn Thomas

2026-05-02

Figure 1: Two test-tube racks side by side on a clean lab bench, the right rack noticeably smaller and holding two amber-tinted tubes, symbolising the contraction in scaffolding when a suite moves from testthat to tinytest.

Same coverage, fewer lines: the appeal of tinytest is less about novelty and more about the absence of ceremony.

Introduction

I did not really appreciate how much ceremony `testthat` carries until I cracked open `inst/tinytest/` in a small package and saw a fourteen-line file doing the work of an eighty-line `test_that()` file in the package I had been maintaining. The expectations were the same. The fixtures were the same. What was missing was the nesting, the helper boilerplate, and roughly half of the imports.

What I discovered, once I started porting the suite, was that `tinytest` is not 'testthat lite'. It is a different shape. The unit of testing is the file, not the `test_that()` block. Tests are flat

assertions evaluated in a script, not nested calls inside a closure. There is no `setup()` and no implicit fixture caching. Several `testthat` idioms (`expect_snapshot`, `local_edition`, `expect_message` with regex matching) have no direct equivalent and require either a different pattern or, in a few cases, a deliberate decision to drop the test.

We walk through the conversion as a paired-example review. I cover layout differences, the mechanical mapping of `expect_*` calls, the handful of patterns that do not port cleanly, the CI implications, and a short list of cases where I concluded that staying on `testthat` was the right call. The companion deliverable, a side-by-side conversion recipe, lives at `docs/testthat-to-tinytest-recipe.qmd` in this post's compendium.

Pros and cons at a glance

Before getting into the mechanical details, a level-set on what each framework does well and where it asks something of the package author. This is an extended summary so that readers can calibrate whether the conversion is worth their afternoon.

What `testthat` does well

- **A mature ecosystem.** The framework is the de-facto standard on CRAN, which means the largest body of Stack Overflow answers, blog posts, and worked examples points at testthat idioms. Onboarding contributors who already write R is faster with testthat than with anything else.
- **Snapshot testing.** `expect_snapshot()`, `expect_snapshot_value()`, and `expect_snapshot_file()` cover text, structured-data, and binary outputs respectively, with automatic regeneration on review. This is genuinely useful for packages whose output is a formatted table, a printed object, or a plot.
- **Rich diff output.** Failed assertions are reported through `waldo`, which produces a side-by-side diff for vectors, lists, and data frames. The diff often points directly at the responsible value, where a bare 'not equal' message would not.
- **IDE integration.** RStudio's test pane reports per-test pass/fail status with one-click navigation. `Ctrl-Shift-T` runs the suite from the editor.
- **Parallel test execution.** `testthat` 3.0+ runs tests in parallel with `parallel = TRUE` in `testthat::test_local()` or `Config/testthat/parallel: true` in DESCRIPTION. For suites with long per-file fixtures, the speed-up is noticeable.
- **Editions for managing breaking changes.** The `Config/testthat/edition: 3` declaration locks the package to a specific behavioural contract, which makes upgrades predictable. testthat 4 will likely use the same mechanism.
- **A wider assertion surface.** `expect_s3_class`, `expect_s4_class`, `expect_type`, `expect_named`, `expect_no_error`, `expect_no_warning`, `expect_no_message`, `expect_setequal`, `expect_mapequal`, and `expect_invisible` all have semantics that read clearly at the call site.
- **`test_check()` exposes the package internal namespace.** Test files reference non-exported helpers and constants by their bare names. For internal-heavy packages, this removes a layer of `getFromNamespace()` ceremony.
- **Helper files auto-sourced.** Files matching `tests/testthat/helper-*.R` are sourced before each test file, providing implicit fixture setup.

- **Skip helpers for environment conditions.** `skip_on_cran()`, `skip_on_os()`, `skip_if_offline()`, `skip_if_not_installed()` are concise and read like the intent they encode.

Where `testthat` asks something of the author

- **A heavy dependency surface.** Installing `testthat` pulls in `waldo`, `brio`, `desc`, `pkgload`, `processx`, `cli`, `digest`, `evaluate`, `R6`, `rlang`, and roughly two dozen transitive dependencies. For toy packages or small internal tools, this is most of the package's install footprint.
- **Slower CI.** A `R CMD check` of an empty `testthat` suite takes longer than the same check with `tinytest` because the framework itself takes time to load and dispatch.
- **The framework itself depends on third-party packages.** This is a philosophical concern more than a practical one, but it does mean a bug or breaking change in any of the transitive dependencies can affect the test suite.
- **Edition complexity.** Editions are useful but add a layer of indirection. Tests that pass under edition 2 may fail under edition 3 (and vice versa). Diagnosing these edition-driven regressions is rare but bewildering when it happens.
- **Per-test ceremony.** Every assertion lives inside a `test_that('description', { ... })` block. The description string is sometimes informative and sometimes ceremonial.

What `tinytest` does well

- **Zero `Imports`.** `tinytest` is a single source file with no third-party dependencies. The framework is itself testable from a fresh R installation.
- **Faster `R CMD check`.** The framework loads in milliseconds. For a small package with a few hundred assertions, the wall-clock saving over `testthat` is on the order of seconds, not minutes, but it compounds across a CI fleet.
- **File-as-test-unit.** Each `inst/tinytest/test_*.R` file is the unit. The natural one-to-one map between an R/ source file and a test file makes the suite easy to navigate. Tests inside a file are top-level, not nested.
- **Tests installed with the package.** Because tests live in `inst/tinytest/`, they ship with the installed package and remain runnable via `tinytest::test_package('pkg')` after install. Users can verify their installation; this is useful for compiled-code packages and packages with system dependencies.
- **A smaller cognitive surface.** No editions, no fixture caching, no helper auto-sourcing, no description strings. When a test fails, the failure mode is straightforward. When it passes, one can be confident it passed for the reason intended.
- **Predictable behaviour across R versions.** With no third-party dependencies, `tinytest` is unaffected by changes in the broader package ecosystem. A test suite written against `tinytest` 1.4 in 2024 will still run unchanged in 2030.
- **Self-evident diagnostics.** A `tinytest` failure reports the file, line, expected value, and actual value with no framework noise. The diagnostic surface is small enough to read at a glance.

Where `tinytest` asks something of the author

- **No snapshot testing.** This is the largest functional gap. Manual reference-file comparison via `capture.output()`
  - `expect_equal()` works but is ergonomically distant from `expect_snapshot()`.
- **No parallel execution by default.** A `cl` argument to `test_all()` accepts a `parallel` cluster, but the default is serial.
- **No RStudio test-pane integration.** Tests run under `R CMD check` and the console; the IDE's 'Run Tests' button does not populate `tinytest` results.
- **A smaller user community.** Searches for specific assertion patterns yield fewer results, and answers are sometimes outdated. The CRAN vignette is the most reliable reference.
- **A narrower assertion surface.** No `expect_s3_class`, `expect_type`, `expect_named`, or `expect_no_error`. Tests using these need manual rewriting; the recipe section 12 documents the substitutions.
- **`test_package()` only sees exports.** Tests that reference non-exported package internals must qualify the reference (`getFromNamespace()` or `pkg:::name`) or the package must export the object. Recipe section 20 has the detail.
- **No `helper-*.R` auto-sourcing.** Shared fixtures require explicit `source('helper_X.R', local = TRUE)` at the top of each consumer file.
- **No `skip_on_os`, `skip_on_ci`, `skip_if_offline` helpers.** The substitutes (`if (... condition ...) exit_file('reason')` at file scope) are equally clear but more verbose at the use site.
- **No `patrick`-style parameterised tests.** Plain `for` loops work, but the failing-iteration identifier is the line number rather than a parameter label.
- **Less informative diff output.** Without `waldo`, large vector or list comparisons report 'not equal' without the side-by-side diff. For complex objects, this lengthens debugging time.

Decision matrix

If your package…	Likely better fit
Has more than five `expect_snapshot_*()` calls	testthat
Is developed primarily in RStudio with the test pane	testthat
Uses `parallel = TRUE` for acceptable CI time	testthat
Has a large team unfamiliar with `tinytest`	testthat
Is a small utility with no snapshots	tinytest
Is tested mainly in CI rather than interactively	tinytest
Wants minimal install footprint	tinytest
Ships compiled code that needs post-install verification	tinytest
Has tests that primarily reference exported functions	either
Is greenfield	either

The matrix is not exhaustive, and most packages have at least one row pulling each direction. The recommendation is not to choose by majority vote but to identify the row that costs the most under the wrong choice. For a package whose CI is too slow, that row is the install-footprint one. For a package whose IDE workflow is critical, the test-pane row dominates.

Motivations

- Reading the `testthat` source after a debugging session and realising how much of the dependency surface (`waldo`, `brio`, `desc`, `pkgload`, `processx`, `cli`) my small packages were pulling in only to call `expect_equal` half a dozen times.
- A growing collection of toy packages where `R CMD check` time was dominated by test-framework startup rather than the tests themselves, particularly inside containers.
- Curiosity about Mark van der Loo's design argument that a test framework should itself be testable, and that a single-file, zero-dependency framework is more honest about what testing actually requires.
- The ergonomic appeal of file-as-test-unit when most of my packages have one source file per concern; the one-to-one map between `R/foo.R` and `inst/tinytest/test_foo.R` makes the suite easy to navigate.
- A practical interest in seeing what breaks. Several `testthat` idioms are deeply embedded in the broader R-package ecosystem (snapshot testing, parallel test runners, RStudio's test pane). Porting forces a clear-eyed audit of what one actually uses.

Objectives

By the end of the conversion, the package should have:

1. An `inst/tinytest/` directory containing one test file per source-code concern, each runnable in isolation via `tinytest::run_test_file()`.
2. A `tests/tinytest.R` bootstrap that triggers the suite during `R CMD check`, replacing the `tests/testthat.R` bootstrap.
3. A `DESCRIPTION` file with `tinytest` in `Suggests`, no `Config/testthat/edition` line, and no `testthat` dependency.
4. A CI run (locally or in GitHub Actions) that exercises the suite end-to-end and reports the same number of distinct assertions the `testthat` suite covered, less any tests that were intentionally retired.

Figure 2: A cappuccino in a dark cup resting on an open paper planner, lit by a shaft of light from a window above. Placeholder ambiance image; will be replaced with subject-specific imagery before publication.

What is tinytest?

`tinytest` is a single-file R testing framework, written by Mark van der Loo, that ships with no `Imports` and one test file per package concern. Where `testthat` organises tests as nested calls inside a `test_that()` closure, `tinytest` evaluates each file as a script and treats every top-level `expect_*` call as an independent assertion.

A useful analogy: `testthat` is to `tinytest` as a full-featured build system is to a Makefile. Both produce the same artifact; one is opinionated about structure, the other gets out of the way.

A concrete example. The same assertion in both frameworks:

```
# testthat
test_that('addition is associative', {
  expect_equal((1 + 2) + 3, 1 + (2 + 3))
})

# tinytest
expect_equal((1 + 2) + 3, 1 + (2 + 3))
```

The `tinytest` form is not nested in anything. The file in which it lives is the unit. The label, when reporting, is the file name plus the line number. There is no descriptive string parameter because the assertion is its own description.

Prerequisites

- R 4.1 or later (the native pipe is used in the post's worked examples, but `tinytest` itself works back to R 3.5).
- An existing R package with a `tests/testthat/` directory. Greenfield packages can skip the conversion entirely and start with `tinytest`.
- Familiarity with `R CMD check` and the standard `tests/<framework>.R` bootstrap convention.
- Optional but useful: a CI configuration file you control. The examples below use GitHub Actions, but the pattern is the same for GitLab, Jenkins, or a local Makefile.

Installation

`tinytest` is on CRAN, with no compiled code and no dependencies. Install it from R:

```
install.packages('tinytest')
```

Verify the install:

```
packageVersion('tinytest')
# [1] '1.4.1'
```

For a package that uses `renv`, add the dependency to the lockfile:

```
renv::install('tinytest')
renv::snapshot()
```

Layout: testthat vs tinytest

The two frameworks place test files in different directories, and the bootstrap files differ as well. The package layout shifts as follows:

```
Before                          After
------                          -----
tests/                          tests/
  testthat.R                      tinytest.R
  testthat/                     inst/
    test-foo.R                    tinytest/
    test-bar.R                      test_foo.R
    helper-fixtures.R               test_bar.R
                                    helper_fixtures.R
```

Three differences worth naming:

1. **Test files move from `tests/testthat/` to `inst/tinytest/`.** The `inst/` location means the tests are installed alongside the package and remain runnable after install via `tinytest::test_package('mypkg')`. This is one of the larger conceptual shifts: in `tinytest`, tests are part of the shipped package, not a development-only artefact.
2. **File-naming convention changes from `test-*.R` to `test_*.R`.** The underscore is `tinytest`'s default; the dash also works, but the underscore is what `tinytest::test_package()` expects without configuration.
3. **The bootstrap file changes name and content.** See below.

The bootstrap file

`tests/testthat.R` looks like this:

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

The `tinytest` equivalent is `tests/tinytest.R`:

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

The `requireNamespace` guard is idiomatic for `tinytest`. Because `tinytest` lives in `Suggests`, not `Imports`, the package should build and check on a system where `tinytest` is not installed. The guard makes that case a no-op rather than a hard error.

The mechanical conversion

The bulk of the work is rewriting individual test files. The mapping is mostly one-to-one. The full table lives in the companion deliverable; here are the patterns that come up most often.

Pattern 1: a 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.

Pattern 2: multiple `test_that()` blocks per file

Before:

```
test_that('addition works', {
  expect_equal(1 + 1, 2)
})

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

After (option A, single file):

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

After (option B, split into two files):

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

Option B is the more idiomatic `tinytest` choice when the two groups exercise unrelated source files. The framework offers no mechanism for grouping inside a file, so cohesion is enforced by file boundaries instead.

Pattern 3: shared fixtures

`testthat` resolves shared fixtures via files named `helper-*.R`, which it sources before each test file. `tinytest` does not have an equivalent automatic mechanism, but two patterns work:

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

```
# inst/tinytest/test_summary.R
source('helper_fixtures.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. The repetition (`source(...)` at the top of each consumer file) is deliberate: `tinytest` privileges explicit dependency over magic.

Pattern 4: skip-on-CRAN

Before:

```
test_that('integration test', {
  skip_on_cran()
  skip_if_not_installed('database_driver')
  ...
})
```

After:

```
if (!identical(Sys.getenv('NOT_CRAN'), 'true')) exit_file('Skipping on CRAN')
if (!requireNamespace('database_driver', quietly = TRUE)) {
  exit_file('database_driver not installed')
}
...
```

`exit_file()` halts the current file with the supplied message recorded as a skip. There is no `tinytest` analogue of `skip_on_os()` or `skip_if_offline()`, but those checks are small enough that an `if (...) exit_file(...)` line at the top of the file is sufficient.

Pattern 5: snapshots

This is the pattern that does not port. `testthat` snapshots (`expect_snapshot()`, `expect_snapshot_value()`, `expect_snapshot_file()`) have no direct equivalent in `tinytest`. The available substitutes are:

- For text output, capture with `capture.output()` and compare to a reference string with `expect_equal()`.
- For binary or large outputs, write a reference file once and compare with `tools::md5sum()` on each run.
- For images, use `tinytest::expect_equal_to_reference()` or drop the test.

In practice, snapshot-heavy suites are the strongest argument against converting. The migration cost is high, the failure modes are subtler than `expect_equal()`, and the resulting tests are noticeably less ergonomic than their `testthat` counterparts.

DESCRIPTION changes

The package metadata changes in three places:

```
Before                            After
------                            -----
Suggests:                         Suggests:
    testthat (>= 3.0.0)               tinytest
Config/testthat/edition: 3        (line removed)
```

If `testthat` is the only test framework being dropped, the removal is straightforward. If the package previously declared both, leave whichever frameworks are still in use.

CI: GitHub Actions

The standard `r-lib/actions/check-r-package` action runs `R CMD check`, which exercises whichever bootstrap file is in `tests/`. No CI changes are strictly required: a package with `tests/tinytest.R` will run its `tinytest` suite during `R CMD check` exactly as a `testthat` package does.

The one wrinkle is that `tinytest` errors are currently surfaced through `R CMD check` as ordinary stop conditions, not as a separate test report. If the CI surface relies on a structured test summary (for example, GitHub's check-run output), an extra step that calls `tinytest::test_package()` directly and emits a TAP report may be useful:

```yaml
- name: Run tinytest with TAP output
  run: |
    Rscript -e 'tinytest::test_package("mypkg",
      side_effects = TRUE) |> as.data.frame() |> print()'
```

Verification

After the conversion, three commands should produce output consistent with the previous `testthat` run:

```
# 1. Bootstrap runs the suite under R CMD check
R CMD check --as-cran .

# 2. Direct invocation reports per-file pass/fail
Rscript -e 'tinytest::test_package(".")'

# 3. Single-file run, useful during development
Rscript -e 'tinytest::run_test_file(
  "inst/tinytest/test_arithmetic.R")'
```

The total number of assertions reported in step 2 should match the `testthat` baseline, less any tests retired during the migration. If the count differs and the difference is not explained by an explicit retirement, a `test_that()` block likely contained more `expect_*` calls than the rewrite captured.

Daily Workflow

Action	Command
Run the full suite	`Rscript -e 'tinytest::test_package(".")'`
Run a single file	`tinytest::run_test_file('inst/tinytest/test_X.R')`
Run all files in a directory	`tinytest::run_test_dir('inst/tinytest')`
Run during `R CMD check`	`R CMD check .` (uses `tests/tinytest.R`)
Build a coverage report	`covr::package_coverage()` (works with both)
Continuous run on file change	`tinytest::test_package('.', side_effects = TRUE)`

Figure 3: An espresso cup viewed from above on a wide hardwood floor, low-key lighting, single subject in deep shadow. Placeholder ambiance image; will be replaced with subject-specific imagery before publication.

Things to Watch Out For

These are the gotchas that cost me time during the conversion. Several are not in the `tinytest` documentation because they are artefacts of the migration rather than the framework itself.

1. **Helper files are not auto-sourced.** A `testthat` package with `tests/testthat/helper-data.R` gets that file sourced before every test file automatically. `tinytest` does not. Symptom: `object 'make_test_data' not found`. Fix: add `source('helper_fixtures.R', local = TRUE)` at the top of each consumer file.

2. **`expect_message()` and `expect_warning()` regex matching.** `testthat` accepts a regex as the second argument. `tinytest` does not. Symptom: tests pass that should not, because any message satisfies the assertion. Fix: assert the message content separately:

   ```
   m <- capture.output(my_function(), type = 'message')
   expect_true(grepl('expected pattern', m))
   ```

3. **`expect_error()` returns the error invisibly.** Capturing it for further inspection requires `e <- expect_error(my_function())` which is the same idiom as `testthat`, but the error class is reported differently. `testthat`'s `class =` argument has no `tinytest` analogue; inspect `class(e)` manually.

4. **`Config/testthat/edition: 3` lingers.** Forgetting to remove this line from `DESCRIPTION` is harmless but confusing for future maintainers. Symptom: a `DESCRIPTION` that references a framework the package no longer uses. Fix: remove the line manually; no tooling performs this step automatically.

5. **Snapshot tests have no clean port.** Listed above as pattern 5; worth re-stating. If a substantial fraction of the suite is `expect_snapshot_*()`, reconsider the migration.

6. **The RStudio test pane does not light up.** `tinytest` integrates with RStudio's build pane through `R CMD check` but does not populate the test runner pane the way `testthat` does. Symptom: 'Run Tests' button in the IDE no longer reports per-test results. Fix: run from the console or terminal; the IDE integration is unlikely to change.

7. **Parallelism requires explicit setup.** `testthat` 3.0+ parallelises tests with `parallel = TRUE`. `tinytest` does not parallelise out of the box, though `tinytest::test_all()` accepts a `cl` argument for a `parallel` cluster. Suites with long-running per-file setup may run noticeably slower in serial.

8. **Four `testthat` 3 assertion functions silently resolve via `pkgload`.** A real conversion of the `zzpower` package surfaced this: `expect_s3_class`, `expect_type`, `expect_named`, and `expect_no_error` are exported by `testthat` but not by `tinytest`. When a developer runs the converted suite via `pkgload::load_all()` followed by `tinytest::run_test_dir()`, `pkgload` auto-attaches every package in `Suggests`, including `testthat`. The four testthat assertions then resolve against the testthat namespace, the run reports 'all ok', and the migration appears complete. Symptom: the tinytest assertion count silently undercounts the testthat baseline. Fix: detach `testthat` before running the suite, or grep the converted files for these four function names and replace them with tinytest equivalents (`expect_inherits`, `expect_true(is.function(...))`, `expect_equal(sort(names(x)), sort(...))`, and unwrapping the `expect_no_error` block). Recipe section 12 has the full mapping.

9. **Non-exported objects are invisible to `tinytest::test_package()`.** `testthat::test_check()` runs tests with the package's internal namespace exposed, so test files can reference non-exported helpers, constants, and S3 methods by their bare names. `tinytest::test_package()` calls `library(pkg)`, which only attaches exports. A test file that worked under testthat with `expect_true(exists('MY_CONST'))` will fail under tinytest with 'object 'MY_CONST' not found', even though the original suite passed. Symptom: under `pkgload::load_all() + tinytest::run_test_dir()`, tests pass; under `R CMD check` (which uses `tinytest::test_package()` against the installed package), the same tests fail. Fix: either export the object via roxygen2 (`#' @export`) and re-run `devtools::document()`, or qualify the reference in the test using `getFromNamespace('MY_CONST', 'pkg')` or `pkg:::MY_CONST`. The qualified-reference variant is the safer choice for objects that should remain internal.

10. **`R CMD check` on a source directory does not auto-derive `Author` and `Maintainer` from `Authors@R`.** Under R 4.5.3 on macOS, `R CMD check <directory>` fails with 'Required fields missing or empty: Author Maintainer' when the DESCRIPTION declares only `Authors@R`. The same DESCRIPTION passes when checked via `R CMD build` followed by `R CMD check <tarball>`. The cause is in `tools:::.read_description()`, which reads the DCF file without injecting derived fields. Symptom: a check that was working last month suddenly errors at the very first step. Fix: prefer the canonical workflow (`R CMD build .` then `R CMD check pkg_X.Y.Z.tar.gz`); if direct source-dir checking is required, add explicit `Author:` and `Maintainer:` lines to DESCRIPTION alongside `Authors@R`.

11. **`skip_if_not()` inside `test_that()` was scoped to one test; a flat conversion may halt the entire file.** A naive textual rewrite of `test_that('memory test', { skip_if_not(...); ... })` produces a top-level `if (!cond) exit_file(...)` followed by the body. `exit_file()` halts the entire test file, but in the original the skip applied only to that one `test_that` block; subsequent blocks ran normally. Symptom: the converted suite passes but reports significantly fewer assertions than the testthat baseline, with one entire file showing zero results. Fix: replace the file-level `exit_file()` with a block-level `if (cond) { ... }` wrapper around just the assertions that depend on the condition. Recipe section 6 now documents both forms; pick the one that matches the original scope.

12. **Argument-name differences in `expect_error` and `expect_warning`.** testthat accepts both `regex =` and `regexp =` as the named-argument form for the matching pattern. tinytest accepts only `pattern =`. Symptom: `unused argument (regex = "...")` or `unused argument (regexp = "...")` errors during the tinytest run. Fix: rename both to `pattern =`. The recipe now lists this in section 12.

Uninstall / Rollback

If the conversion turns out to be a poor fit (snapshot-heavy suite, IDE-dependent workflow, parallelisation requirements), the rollback is straightforward because the original `tests/testthat/` directory has not been deleted. The git history preserves it.

```
# 1. Restore tests/testthat/ from git
git checkout HEAD~N -- tests/testthat tests/testthat.R

# 2. Remove the tinytest scaffolding
```

```
rm -rf inst/tinytest tests/tinytest.R

# 3. Restore DESCRIPTION
git checkout HEAD~N -- DESCRIPTION
```

The `HEAD~N` placeholder is whatever commit preceded the migration. The cleaner approach is to perform the conversion on a branch and revert the branch if needed, rather than committing the migration to `main`.

Figure 4: A close-up of a single espresso drip falling from a machine into a grey ceramic mug. Placeholder ambiance image; will be replaced with subject-specific imagery before publication.

What Did We Learn?

Lessons Learnt

Conceptual.

- The unit of testing is a design choice, not a given. `testthat` treats the assertion as the unit and groups assertions into `test_that()` closures. `tinytest` treats the file as the unit and treats assertions as top-level statements. Neither is strictly better; the choice has consequences for fixture scope, error reporting, and how new tests are added.
- A test framework's dependency surface matters more than the number of features it offers. A package whose tests pull in forty transitive dependencies pays an installation cost on every CI run, whether or not the features are used.
- 'Snapshot testing' is a separable concern from unit testing. Treating the two as a single problem (which `testthat` does via `expect_snapshot_*()`) makes the framework richer but also more entangled. `tinytest` declines to merge them, which forces a deliberate choice about whether snapshots belong in the test suite at all.
- Assertions read more naturally when they are not nested inside a description string. The test-name parameter that `test_that()` requires is information that the file name and line number carry implicitly; making it explicit is helpful in some cases and ceremonial in others.

Technical.

- `tinytest::expect_equal()` defaults to `tolerance = 1e-6`, matching `base::all.equal()` rather than `testthat::expect_equal()`'s newer default of strict equality with a configurable tolerance. Tests that previously passed under `testthat` 3 may need an explicit `tolerance = 0` on the `tinytest` side, or vice versa.
- The order in which test files run matters more than under `testthat`. `tinytest` evaluates files in alphabetical order by default. If a file has side effects on the global state (writes to disk, changes options, alters the working directory), later files will see them.
- `tinytest::test_package()` returns a `tinytests` object whose `as.data.frame()` method gives a per-assertion table. This is a more programmatic interface than `testthat::test_local()`'s console-output focus, and pairs well with custom CI summaries.
- Coverage tools (`covr`) work without modification. The package introspection that `covr` performs is framework- agnostic; it instruments source files rather than test files.

Gotchas.

- Forgetting to rename `helper-*.R` to `helper_*.R` (dash vs underscore). `tinytest` does not source files starting with `helper`; the convention is purely human-facing, so a typo here is silent.
- Treating `inst/tinytest/` as private. Because tests live in `inst/`, they are installed with the package and visible to users. This is a feature (users can run the suite to verify their installation) but it means the test files should not contain credentials, paths to private servers, or other artefacts that do not belong in a published package.
- Mixing `testthat` and `tinytest` during a partial migration. Both bootstraps will run during `R CMD check`. If the same assertion is duplicated, the count doubles; if it is split, the report fragments. Pick one and finish before merging.

- Underestimating the snapshot-test inventory. Run `grep -r 'expect_snapshot' tests/testthat/` before starting. If the count is non-trivial, the migration will be longer than expected.

Limitations

- `tinytest` lacks built-in snapshot testing, parallel test execution, and IDE integration with RStudio's test pane. For packages that depend on any of these, the migration is a net loss.
- The framework's spartan design means that some `testthat` affordances (parameterised tests via `patrick`, fixtures via `withr::local_*`) require manual reimplementation.
- Error messages in `tinytest` are less informative by default than `testthat` 3's `waldo`-powered diffs. For numeric or list comparison, the diff output is the single feature `tinytest` users miss most.
- The framework has a small user community relative to `testthat`. Searching for solutions to specific assertion patterns yields fewer results, and the available answers are sometimes outdated.

Opportunities for Improvement

1. Build a `tinytest` linter that flags accidental `testthat` idioms (`test_that()` calls, `skip_on_*()`, regex arguments to `expect_message`) during the migration.
2. Wrap the common conversion patterns in a small package (`tinyport`?) that automates the file-rename, header rewrite, and DESCRIPTION update steps.
3. Add a `tinytest`-aware action to GitHub's R-lib actions collection so that the structured test-summary step described above is one click away.
4. Document the snapshot-equivalent patterns more thoroughly. The CRAN vignette covers the basics; a longer reference that addresses image, JSON, and structured-output snapshots would close a real gap.
5. Build a per-package decision script that reports the number of `expect_snapshot_*()` calls, the number of `helper-*.R` files, and an estimated hour-cost for the migration.
6. Contribute a `parallel = TRUE` argument to `tinytest::test_package()` that mirrors `testthat`'s built-in parallelisation.

Wrapping Up

The conversion is a small project, on the order of an afternoon for a package with a few hundred assertions and no snapshots. The mechanical work is straightforward and the patterns repeat. What takes the time is the audit: deciding which `testthat` features the package actually uses, which can be replaced, and which signal that the migration is the wrong call.

The decision matrix is short. If the package uses snapshot tests heavily, depends on RStudio's IDE integration, or relies on parallel test execution, stay on `testthat`. If the package is small, has no snapshots, and is tested in CI rather than interactively, the migration is mostly upside: fewer dependencies, faster `R CMD check`, a smaller cognitive surface.

This post stops short of recommending the conversion as a default. Both frameworks are well-engineered. The choice between them is an architectural decision about how much ceremony a test framework should impose, and that decision is package-specific.

Wrapping Up

In conclusion, six points merit emphasis. First, the mechanical mapping is mostly one-to-one; snapshots are the exception. Second, the `inst/tinytest/` placement makes tests installable, which is the largest conceptual shift. Third, helper files are not auto-sourced, so explicit `source()` calls are required. Fourth, `Config/testthat/edition` lingers in `DESCRIPTION` if the line is not removed manually. Fifth, the migration is reversible; performing it on a branch is the recommended practice. Sixth, snapshot-heavy suites are a strong reason to remain on `testthat`.

See Also

- `tinytest` on CRAN: https://cran.r-project.org/package=tinytest
- van der Loo, M. (2020). 'tinytest: A Lightweight and Feature Complete Unit Testing Framework for R Packages'. CRAN vignette.
- `testthat` documentation: https://testthat.r-lib.org/
- Post 40 (testing for data analysis workflow), in this blog, for the broader context of testing in compendium-style R projects.
- Post 61 (zzcollab analysis checklist), in this blog, for an example of `tinytest` and `testthat` used side-by-side in a single project.

Reproducibility

Component	Version	Notes
OS	macOS 15.4	also tested on Ubuntu 24.04
R	4.4.1	matches `renv.lock`
`tinytest`	1.4.1	from CRAN, 2024-02 release
`testthat`	3.2.1	reference baseline
`covr`	3.6.4	optional, used for coverage
Date verified	2026-05-02	claims in this post tested on this date

```
sessionInfo()
```

Appendix A: Conversion Atlas

A condensed mapping of the most common `testthat` idioms to their `tinytest` equivalents. The full table, with paired examples for each row, is in `docs/testthat-to-tinytest-recipe.qmd` in this post's compendium.

testthat	tinytest
`expect_equal(x, y)`	`expect_equal(x, y)`
`expect_identical(x, y)`	`expect_identical(x, y)`
`expect_true(x)`	`expect_true(x)`
`expect_false(x)`	`expect_false(x)`
`expect_error(f(), 'msg')`	`expect_error(f(), pattern = 'msg')`
`expect_warning(f(), 'msg')`	`expect_warning(f(), pattern = 'msg')`
`expect_message(f(), 'msg')`	`expect_message(f(), pattern = 'msg')`
`expect_null(x)`	`expect_null(x)`
`expect_silent(f())`	`expect_silent(f())`
`expect_snapshot(x)`	(no direct equivalent; see pattern 5)
`expect_s3_class(x, 'cls')`	`expect_inherits(x, 'cls')`
`expect_is(x, 'cls')`	`expect_inherits(x, 'cls')`
`expect_type(x, 'closure')`	`expect_true(is.function(x))`
`expect_type(x, 'double')`	`expect_equal(typeof(x), 'double')`
`expect_named(x, c(...))`	`expect_equal(sort(names(x)), sort(c(...)))`
`expect_no_error({block})`	(unwrap; inner assertions stand on their own)
`expect_lt(x, y)`	`expect_true(x < y)`
`expect_gt(x, y)`	`expect_true(x > y)`
`expect_lte(x, y)`	`expect_true(x <= y)`
`expect_gte(x, y)`	`expect_true(x >= y)`
`expect_error(f(), regex = 'msg')`	`expect_error(f(), pattern = 'msg')`
`expect_warning(f(), regexp = 'msg')`	`expect_warning(f(), pattern = 'msg')`
`skip_on_cran()`	`if (!nzchar(Sys.getenv('NOT_CRAN')))` `exit_file('CRAN')`
`skip_if_not_installed('pkg')`	`if (!requireNamespace('pkg', quietly = TRUE))` `exit_file('pkg missing')`
`skip_on_ci()`	`if (nzchar(Sys.getenv('CI'))) exit_file('CI')`
`skip_if_not(cond, msg)` (per-test)	`if (cond) { ... }` (block-level wrapper)
`context('label')`	(remove; tinytest has no equivalent)
`library(testthat)` in test files	(remove; tinytest is implicit at runtime)
`helper-foo.R` (auto-sourced)	`source('helper_foo.R', local = TRUE)`

Appendix B: A Real Migration on `zzpower`

The recipe was validated on `zzpower` v0.3.0, an interactive Shiny power-analysis package, on branch `testthat-to-tinytest`. Pre-migration: 10 testthat files, 1,734 LOC, 602 expectations passing under `testthat::test_dir()`. Post-migration: 10 tinytest files, 1,625 LOC, 600 assertions passing under `tinytest::run_test_dir()` with `testthat` detached.

The 6% line-count reduction (109 lines) came almost entirely from removing `test_that()` wrappers and de-indenting bodies by two spaces. The 2-assertion delta is exactly the two `expect_no_error()` outer-wrapper counts that no longer count as assertions in their own right; the inner expectations they guarded are still counted.

The migration was not entirely mechanical. Four `testthat` 3 assertion functions had no direct port and required manual substitution: `expect_s3_class` (9 calls; replaced with `expect_inherits`), `expect_type` (3 calls; replaced with `expect_true(is.function(...))` for the closure case and `expect_equal(typeof(...), ...)` otherwise), `expect_named` (1 call; replaced with `expect_equal(sort(names(x)), ...)`), and `expect_no_error` (2 calls; the wrapper was removed). These four patterns were not in the recipe's first draft, because the conversion script (which wrapped a textual brace-aware transform around `test_that()` calls) preserved them verbatim. The 'Things to Watch Out For' section now includes these patterns as gotcha 8, and the recipe has a new section 12 with paired examples.

A subtler trap surfaced during dry-running. `pkgload::load_all()` auto-attaches every package in `Suggests`, which means the testthat namespace becomes available even after the migration removes testthat from the bootstrap. The four testthat-only assertion functions then resolve silently against testthat's exports, the run reports 'all ok', and the migration appears complete. The honest dry-run requires either detaching testthat explicitly or removing it from `Suggests` before verification. Only after `tinytest::run_test_dir()` reports a result count consistent with the testthat baseline (less the known `expect_no_error` deltas) is the migration validated.

`R CMD check` was attempted in two configurations and surfaced two more issues, neither in the recipe's first draft.

The first configuration, `R CMD check <source-directory>`, failed at the very first step with 'Required fields missing or empty: Author Maintainer'. Investigation showed that this is not a parsing issue but a behavioural one: `tools:::.read_description()`, which `R CMD check` calls to load the DESCRIPTION, does not auto-derive `Author` and `Maintainer` from `Authors@R`. The same DESCRIPTION passes when checked via the canonical `R CMD build .` followed by `R CMD check pkg_X.Y.Z.tar.gz`. The lesson generalises: prefer the build-then-check workflow; if direct source-directory checks are required, declare `Author` and `Maintainer` explicitly. Recipe section 17 and gotcha 10 now document this.

The second configuration, `R CMD check <tarball>`, passed the DESCRIPTION step but failed inside `tests/tinytest.R` with 'could not find function "build_sample_size_inputs"'. The cause is that `tinytest::test_package('zzpower')` calls `library('zzpower')`, which only attaches exported objects. zzpower's `test_framework.R` referenced six non-exported helpers (`build_advanced_settings`, `build_effect_size_inputs`, `build_sample_size_inputs`, `get_effect_size_range`, `logrank_power`, `trend_power`) by their bare names, which worked under `testthat::test_check()` but not under tinytest. The fix added six `getFromNamespace()` assignments at the top of `test_framework.R`, bringing the helpers into the test file's scope without modifying the package's NAMESPACE. After the fix, `R CMD check` reported `Status: OK` on the resulting tarball. Recipe section 20 and gotcha 9 document the diagnostic and the fix.

Note that `ZZPOWER_CONSTANTS`, which my first investigation identified as the offender, is in fact exported (`export(ZZPOWER_CONSTANTS)` in the generated NAMESPACE) and accessible through the search path after `library('zzpower')`. The earlier failure was a misdiagnosis from a stale install. The lesson here is operational: when an earlier check-run leaves a stale binary in the test library,

its NAMESPACE may not match the source. Always rebuild and reinstall when chasing 'object not found' errors that contradict a current `getNamespaceExports()` inspection.

Neither issue invalidates the recipe; both refine it. The final state on the validation branch: `R CMD check zzpower_0.4.0.tar.gz` returns `Status: OK`, with all non-skipped checks passing. The total cost of using `zzpower` as the validation case study was higher than expected (roughly four hours rather than the 'half a day to a day' my pre-migration estimate had) because each unanticipated failure required tracing R-internal code to diagnose. The pay-off is that the recipe has been hardened against three real failure modes (the four testthat-only assertion functions, the pkgload-namespace contamination, and the `library` vs `loadNamespace` test- visibility difference) before any reader attempts the migration on their own package.

The migration commit and follow-up commits are on branch `testthat-to-tinytest` of the zzpower repository.

Time spent: roughly two hours, including writing the brace-aware conversion script (about 30 lines of R), running two dry-runs to surface the testthat-namespace contamination issue, and patching the recipe with the four newly-discovered patterns. The migration commit and follow-up commits are on branch `testthat-to-tinytest` of the zzpower repository.

Second case study: `zztable1`

`zzpower` was small enough that the recipe gaps it surfaced might have been idiosyncratic. A second migration on `zztable1` (publication-quality summary tables, version 0.5.0; 13 testthat files, 2,393 LOC, 564 passing assertions plus 1 skipped) exercised the recipe at roughly twice the scale and exposed five additional patterns the first case missed.

Pre-migration, `zztable1` had: 36 `expect_s3_class` calls, 24 `expect_is` calls (testthat's older alias for the same operation), 19 `expect_type` calls, 18 `expect_lt` calls and one `expect_gte`, and several `skip_*` invocations including an in-block `skip_if_not(capabilities('long.double'), ...)` inside a `test_that` block. It also had two condition matchers using the `regex =` and `regexp =` argument names, a `library(testthat)` line in 10 of the 13 files, and a single `context(...)` call.

The migration applied the pattern table in section 12 and the conversion-atlas additions, with the following new findings folded back into the recipe:

1. **Ordering assertions (`expect_lt`, `expect_gt`, `expect_lte`, `expect_gte`) have no tinytest equivalent.** Replace with `expect_true(x < y)` and similar forms. This added 38 substitutions in `zztable1`; the same pattern added zero in `zzpower` because that package's tests happened not to use ordering assertions.
2. **`expect_is` is testthat's deprecated alias for `expect_s3_class`.** It behaves identically; the substitute is `expect_inherits`. zztable1's test suite used both forms inconsistently across files.
3. **`expect_error` and `expect_warning` accept both `regex =` and `regexp =` as the matching-argument name.** tinytest accepts only `pattern =`. Both names need to be renamed.
4. **A `skip_if_not()` call placed inside a `test_that()` block was scoped to that block alone in testthat.** A naive flat conversion places `exit_file()` at the file level, which halts the entire file. This dropped 12 assertions from `test_performance.R` until the conversion was rewritten as a block-level `if (cond) { ... }` wrapper.

5. **`library(testthat)` calls inside test files and bare `context('...')` declarations** are testthat-specific and must be removed from the converted files. Neither surfaces as an error if the testthat package is still installed (because `library(testthat)` succeeds and `context` is exported), but both are noise that will error once testthat is finally removed from `Suggests`.

After applying these substitutions, **564 of 564 expected assertions passed** under `tinytest::test_package('zztable1')`. `R CMD check zztable1_0.5.0.tar.gz` reports `checking tests ... OK`. Three other warnings remain (`build` directory, `dependencies in R code`, `package vignettes`), all stemming from a missing `zzobj2fig` package in `Suggests/Remotes` and unrelated to the migration.

Time spent on `zztable1`: approximately three hours. The incremental cost over `zzpower` came almost entirely from the five new patterns; the brace-aware conversion script itself ran without modification. The recipe has been updated to cover all nine testthat-only assertion functions and the two scoping issues, so a third migration should not turn up additional surprises at this scale.

Appendix C: The `zzedc` Migration (Origins of the Translator)

The recipe in this post was hardened against the `zzedc` electronic data capture package before `zzpower` or `zztable1` were attempted. `zzedc` is the largest package in the portfolio: 51 test files, 989 test blocks, 3,064 assertions. This appendix records why the migration was attempted on that package first, what the Python translator was designed to handle, and the seven-round arc that produced the post-round-7 artifact now in the companion repository.

C.1 When migration is worth the cost

Most R packages should leave `testthat` alone. The framework is well documented, well integrated with `usethis` and `devtools`, and has a much larger user base than any alternative. If a package is already working well on `testthat`, that is the correct choice.

For a smaller class of packages the dependency calculus is different. `testthat` 3 brings roughly thirty transitive packages into a package's `Suggests` field. `tinytest`, by Mark van der Loo, brings zero. The migration becomes worth considering in five concrete contexts:

- CRAN-bound packages where reviewers see and weigh every dependency.
- CI matrices where each row reinstalls `testthat` and its full transitive tail.
- Long-term reproducibility stacks (Docker images, `renv` lockfiles) where a thirty-package surface ages less gracefully than a one-package surface.
- Pharmaverse and FDA-submission contexts, where every dependency appears in a software bill of materials and must be justified.
- Embedded R deployments (AWS Lambda, scratch containers, minimal Posit Connect images) where install size matters.

For `zzedc`, the relevant constraint was the third and fourth: the package targets clinical research environments where the dependency graph is itself a documentation artifact, and the existing research-compendium workflow already exercised `tinytest` elsewhere in the portfolio. If none of these apply, the migration is not worth the disruption.

C.2 The naive approach and why it fails

The temptation is to reach for `sed`. Most `expect_*` names are shared between the two frameworks, and the obvious script handles a useful fraction of cases:

```
sed -E \
  -e 's/library\(testthat\)/library(tinytest)/' \
  -e '/^context\(/d' \
  tests/testthat/test-*.R
```

The result parses. It also fails in at least four substantive ways.

First, `test_that()` has no `tinytest` analogue. The wrapper must be removed. A naive deletion of the wrapping line and its closing brace breaks block scoping: `testthat` runs each block in a fresh function frame, so `on.exit()` fires at the end of the block and local variables do not leak. Top-level tinytest code has neither property.

Second, `skip_if(requireNamespace('haven', quietly = TRUE), 'msg')` contains a comma inside the inner call. A regex that splits at the first comma will mangle the condition. The translator needs a paren-balanced parser, not a regex.

Third, multi-line string literals containing YAML, SQL, or JSON fixtures often use indentation that is part of the string content. An early version of the translator de-indented test bodies to compensate for the dropped wrapper, which silently corrupted those literals.

Fourth, `expect_warning(x, NA)` is `testthat` idiom for ‘expect no warning’. A direct translation passes `NA` to `tinytest::expect_warning` and reports a confusing failure. The correct rewrite is `expect_silent(x)`.

These are not edge cases. All four appeared in the first hundred lines of the `zzedc` test suite.

C.3 What the Python translator handles

The Python translator (`testthat_to_tinytest.py`) handles the patterns below. Each is a fix for a class of failure surfaced during the `zzedc` migration.

Block dropping with scope preservation. `test_that('desc', { ... })` becomes a `# Test: desc` comment plus `local({ ... })` around the body:

```
test_that('creates database', {
  con <- dbConnect(SQLite(), ':memory:')
  on.exit(dbDisconnect(con))
  expect_true(dbIsValid(con))
})
```

becomes

```
# Test: creates database
local({
  con <- dbConnect(SQLite(), ':memory:')
  on.exit(dbDisconnect(con))
  expect_true(dbIsValid(con))
})
```

The `local()` wrapper is the cheapest way to recover testthat's function-frame semantics without rewriting every test.

Nested BDD blocks. `describe(...) { it(...) { ... } }` is handled by iterating the block translator to a fixed point. Each pass strips one layer; ten passes covers everything seen in practice.

Comparison rewrites. `tinytest` does not provide `expect_lt`/`expect_gt`/`expect_lte`/`expect_gte` as first-class expectations. The translator rewrites them to `expect_true(x < y)` and so on. Similarly `expect_length(x, n)` becomes `expect_equal(length(x), n)`, `expect_null(x)` becomes `expect_true(is.null(x))`, `expect_s3_class(x, 'c')` becomes `expect_true(inherits(x, 'c'))`, and `expect_no_error(expr)` becomes `expect_silent(expr)`.

No-warning idiom. `expect_warning(x, NA)` and `expect_warning(x, regexp = NA)` translate to `expect_silent(x)`. The same applies to `expect_error(x, NA)`.

Argument renames. `regexp = '...'` becomes `pattern = '...'` for `tinytest`'s `expect_match` and related functions.

Paren-balanced skip translation. `skip_if(requireNamespace('haven', quietly = TRUE), 'msg')` is parsed as a single call with two top-level arguments. The result is `if ((requireNamespace('haven', quietly = TRUE))) exit_file('msg')`.

Block-scoped skip semantics. `tinytest`'s `exit_file()` aborts the entire file, whereas testthat's `skip_*` aborts only the current block. When a `skip_*` call is the first directive of a `test_that` body, the translator wraps the rest of the block in `if (!cond) local({ ... })` rather than emitting a top-level `exit_file()`. The skip stays scoped to its original block.

C.4 The helper-file consolidation problem

`testthat` auto-loads any file in `tests/testthat/` whose name begins with `helper-`. `tinytest` has no such convention. It does auto-load files in `inst/tinytest/` that begin with an underscore, but only when invoked through `tinytest::test_package()`; running individual files via `tinytest::run_test_file()` does not trigger the auto-load.

The convention used for `zzedc` is to consolidate the four `helper-*.R` files (`helper-test-setup.R`, `helper-skip.R`, `helper-db-backends.R`, `helper-test-utilities.R`) into a single `inst/tinytest/_setup.R`, and to source it explicitly from the top of each translated test file:

```
if (file.exists('_setup.R')) source('_setup.R')
```

This is honest: one large setup file is less ergonomic than four small ones, but `tinytest`'s model is simpler for users to reason about, and the explicit `source()` keeps individual test files runnable without invoking the package-level harness.

The translator emits the `source()` line; the consolidated `_setup.R` is hand-written. The `zzedc _setup.R` runs to about 680 lines, combining database fixtures, multi-backend test harnesses, custom expectations, and `Sys.setenv`/`Sys.unsetenv` lifecycle helpers. The `testthat::skip()` call is reimplemented as a thin wrapper around `exit_file()` so existing helper code continues to compile.

C.5 The seven-round migration arc

The `zzedc` migration proceeded in seven rounds. The arc is a useful empirical record of what a hardened translator does and does not catch on first contact with a real package.

Round 1, naive translator: 0 of 989 tests ran. The first file had a parse error from a YAML literal whose indentation had been stripped by an over-eager de-indent pass; the `tinytest` runner aborts on a single parse failure. Removing the de-indent step and relying on `local({})` for scoping fixed this.

Round 2, after `local({})` wrapping: 981 of 989 tests ran. Eight failures remained.

Rounds 3 through 6 each surfaced one new pattern: paren-balanced skip conditions, block-scoped skip wrapping, the `regexp = NA` idiom, S3 class inheritance, and the custom `expect_table_exists` helper.

Round 7: 3,064 of 3,064 assertions passing.

The eight final failures resolved into two classes. Seven were indentation-stripping inside YAML literals that survived the first de-indent fix because they were nested two levels deep; removing de-indentation entirely (rather than partially) eliminated the class. The eighth was the `expect_warning(x, NA)` semantics, which had been masquerading as a passing test under `testthat` because the regex was `NA` and matched nothing rather than checking warning suppression.

The version of the translator in the companion repository is the post-round-7 artifact.

C.6 Using the tools

For a single file during development, invoke the Python translator directly:

```
python3 testthat_to_tinytest.py \
  tests/testthat/test-mything.R \
  inst/tinytest/test_mything.R
```

The output is a self-contained tinytest file that sources `_setup.R` from its own directory if present.

For a whole repository, the bash orchestrator (`migrate.sh`) handles the surrounding `DESCRIPTION` and scaffold edits. Phase 1 is fully automated for repositories whose tests are three-line `testthat` stubs (the pattern produced by `usethis::use_testthat()` followed by no further work). Phase 2

generates per-file conversion previews into `.migration-previews/` for repositories with substantive test code, which the user reviews and applies manually:

```
./migrate.sh phase1 [--dry]
./migrate.sh phase2
```

Beyond translation, the orchestrator replaces `testthat` with `tinytest` in `DESCRIPTION`, removes `Config/testthat/edition: 3` if present, deletes the `tests/testthat/` tree, writes a new `tests/tinytest.R`, regenerates `renv.lock` from the trimmed `DESCRIPTION`, and commits the result.

After translation, the manual checklist for a real-test repository is:

- Hand-write `inst/tinytest/_setup.R` consolidating any `helper-*.R` files. Replace `library(testthat)` with nothing and `testthat::skip(msg)` with `exit_file(msg)` (or with a wrapped form where block scoping is needed).
- Inspect any `expect_snapshot` and `local_edition` calls; they will not work without manual rewriting.
- Run `tinytest::run_test_dir('inst/tinytest')` and triage the first round of failures.

The translator handles the rote mechanical work: paren-balanced parsing, the seventeen `expect_*` rewrites, the scope-preserving `local({...})` wrapping, the block-scoped skip semantics. It does not replace the judgment required to convert snapshot tests, to consolidate helper files, or to decide which `Suggests` packages should now move to `Imports` because the test suite was the only place using them. That judgment is the actual work of the migration; the translator makes the mechanical part fast enough that the judgment becomes the bottleneck rather than the typing.

Rendered on 2026-05-18 at 09:00 PDT. Source: ~/prj/qblog/posts/62-testthat-to-tinytest/testthat-to-tinytest/analysis/report/index.qmd

---

Let's Connect

If you spotted an error, found a pattern that did not port cleanly, or have a different conclusion about the conversion, the comment thread below is the right place. Issues and pull requests against the post's source are welcome at https://github.com/rgthomas47/qblog.

Related posts in this cluster

This post is part of the *R Package Development and Testing* series. Recommended reading order:

1. Post 70: Updating an R Package: A Complete Workflow
2. Post 72: Writing a Simple Vim Plugin for REPL Interaction
3. Post 73: Testing Data Analysis Workflows in R
4. **Post 74: From testthat to tinytest** (this post)