Package: muttest (via r-universe)

July 21, 2026

Type Package

Title Mutation Testing

Version 0.3.0

Description Measure quality of your tests. 'muttest' introduces small changes (mutations) to your code and runs your tests to check if they catch the changes. If they do, your tests are good. If not, your assertions are not specific enough. 'muttest' gives you percent score of how often your tests catch the changes.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports checkmate, cli, digest, fs, htmltools, jsonlite, mirai, R6, rlang, testthat, tools, treesitter, treesitter.r (>= 1.3.0), withr

Config/testthat/edition 3

URL https://jakubsob.github.io/muttest/

Suggests box, covr, cucumber (>= 2.1.0), ggplot2, jsonvalidate, knitr, purrr, rmarkdown, shiny, stringr

VignetteBuilder knitr

NeedsCompilation no

Author Jakub Sobolewski [aut, cre]

Maintainer Jakub Sobolewski <jakupsob@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-21 12:30:19 UTC

RemoteUrl https://github.com/cran/muttest

RemoteRef HEAD

RemoteSha 5659552aae13ee775c13edef35fa29b356db6d59

Contents

`arithmetic_operators` *Arithmetic operator mutators*

Description

Returns a ready-made list of `operator()` mutators covering common arithmetic swaps: `+`/`-`, `*`/`/`, `^`/`*`, `%%`/`*`, `%/%`/`/`.

Usage

```
arithmetic_operators()
```

Details

Use on any file that performs calculations. A surviving mutant from this preset typically means an assertion checks a *property* of the result (sign, order) rather than a specific value — replacing `expect_gte()` with `expect_equal()` and a computed expected value usually kills it.

Value

A list of `operator()` mutators.

See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the direction-insensitive assertion pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
arithmetic_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/stats.R",
  mutators = arithmetic_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

boolean_literal *Mutate a boolean literal*

Description

Replaces a boolean constant (`TRUE`, `FALSE`, `T`, or `F`) with another one.

Usage

```
boolean_literal(from, to)
```

Arguments

`from`	The literal to be replaced. One of `"TRUE"`, `"FALSE"`, `"T"`, `"F"`.
`to`	The literal to replace with.

Examples

```
boolean_literal("TRUE", "FALSE")
boolean_literal("FALSE", "TRUE")
boolean_literal("T", "F")
boolean_literal("F", "T")
```

boolean_literals *Boolean literal mutators*

Description

Returns a ready-made list of `boolean_literal()` mutators covering all canonical flips: `TRUE`/`FALSE` and `T`/`F`.

Usage

```
boolean_literals()
```

Details

Use on any file that passes or returns boolean flags. A surviving mutant from this preset typically means a test checks a *side effect* of the flag (e.g. the branch taken) rather than the flag value itself — adding `expect_true()`/`expect_false()` on the return value kills it.

Value

A list of `boolean_literal()` mutators.

See Also

vignette("mutators", package = "muttest") for the full mutator table.

vignette("interpreting-results", package = "muttest") for how to diagnose survivors and fix the underlying test weakness.

Examples

```
boolean_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/flags.R",
  mutators = boolean_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

call_name *Mutate a function call name*

Description

Replaces a function name in a call expression with another name. Useful for swapping semantically related functions such as any/all, min/max, or sum/prod.

Usage

```
call_name(from, to)
```

Arguments

from	The function name to replace.
to	The function name to replace with.

Examples

```
call_name("any", "all")
call_name("min", "max")
call_name("sum", "prod")
```

`comparison_operators` *Comparison operator mutators*

Description

Returns a ready-made list of `operator()` mutators covering direction swaps (</>, <=/>=, ==/!=) and boundary shifts (</<=, >/>=).

Usage

```
comparison_operators()
```

Details

Use on any file with threshold logic, range checks, or filter conditions. A surviving mutant from this preset means the exact boundary value implied by the operator was never passed to the function — adding a test at that boundary value kills it.

Value

A list of `operator()` mutators.

See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the missing boundary value pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
comparison_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/shipping.R",
  mutators = comparison_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

`condition_mutations` *Condition mutation mutators*

Description

Returns a ready-made list of `negate_condition()` and `remove_condition_negation()` mutators covering both directions of condition logic: wrapping a plain condition in `!(...)` and stripping `!` from an already-negated condition.

Usage

```
condition_mutations()
```

Details

Use on any file with non-trivial `if`/`while` conditions. Surviving mutants mean the branch outcome was never tested with inputs that cross the boundary — adding a test where the condition flips from `TRUE` to `FALSE` kills them.

Value

A list of mutators.

See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
condition_mutations()

## Not run:
plan <- muttest_plan(
  source_files = "R/validation.R",
  mutators = condition_mutations()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

CopyStrategy *CopyStrategy interface*

Description

Extend this class to implement a custom copy strategy.

Methods

Public methods:

- `CopyStrategy$execute()`
- `CopyStrategy$clone()`

Method `execute()`**:** Copy project files according to the strategy

Usage:

```
CopyStrategy$execute(original_dir)
```

Arguments:

`original_dir` The original directory to copy from

`plan` The current test plan

Returns: The path to the temporary directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CopyStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other CopyStrategy: `PackageCopyStrategy`, `default_copy_strategy()`

default_copy_strategy *Create a default project copy strategy*

Description

Create a default project copy strategy

Usage

```
default_copy_strategy(...)
```

Arguments

...	Arguments passed to the ?PackageCopyStrategy constructor.

Value

A ?CopyStrategy object

See Also

Other CopyStrategy: CopyStrategy, PackageCopyStrategy

---

default_reporter *Create a default reporter*

Description

Create a default reporter

Usage

```
default_reporter(...)
```

Arguments

...	Arguments passed to the ?ProgressMutationReporter constructor.

See Also

Other MutationReporter: JSONMutationReporter, MultiReporter, MutationReporter, ProgressMutationReporter

---

default_test_strategy *Create a default run strategy*

Description

Create a default run strategy

Usage

```
default_test_strategy(...)
```

Arguments

...	Arguments passed to the ?FullTestStrategy constructor.

Value

A ?TestStrategy object

See Also

Other TestStrategy: FileTestStrategy, FullTestStrategy, TestStrategy

---

delete_statement *Delete statements one at a time*

Description

Produces one mutant per deletable statement, removing each `x <- expr` assignment or standalone `f(...)` call from the source. Surviving mutants reveal untested side effects or dead assignments.

Usage

```
delete_statement()
```

Details

Function definitions (`x <- function(...) { ... }`) are left untouched to avoid producing structurally broken mutants.

Value

A Mutator object.

Examples

```
delete_statement()
```

FileTestStrategy *Run tests matching the mutated source file name*

Description

This strategy tells if a mutant is caught by a test matching the source file name.

For example, if the source file name is `foo.R`, and there are test files named `test-foo.R` or `test-bar.R`, only `test-foo.R` will be run.

This strategy should give faster results than ?FullTestStrategy, especially for big codebases, but the score might be less accurate.

Super class

muttest::TestStrategy -> FileTestStrategy

Methods

Public methods:

- `FileTestStrategy$new()`
- `FileTestStrategy$execute()`
- `FileTestStrategy$clone()`

Method `new()`**:** Initialize the FileTestStrategy

Usage:

```
FileTestStrategy$new(
  load_helpers = TRUE,
  load_package = c("source", "none", "installed")
)
```

Arguments:

`load_helpers` Whether to load test helpers

`load_package` The package loading strategy

Method `execute()`**:** Execute the test strategy

Usage:

```
FileTestStrategy$execute(path, plan, reporter)
```

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `muttest_plan()`.

`reporter` The reporter to use for test results

Returns: The test results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FileTestStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FullTestStrategy`, `TestStrategy`, `default_test_strategy()`

`FullTestStrategy` *Run all tests for a mutant*

Description

This test strategy tells if a mutant is caught by any test.

To get faster results, especially for big codebases, use `?FileTestStrategy` instead.

Super class

`muttest::TestStrategy` -> `FullTestStrategy`

Methods

Public methods:

- `FullTestStrategy$new()`
- `FullTestStrategy$execute()`
- `FullTestStrategy$clone()`

Method `new()`**:** Initialize

Usage:

```
FullTestStrategy$new(
  load_helpers = TRUE,
  load_package = c("source", "none", "installed")
)
```

Arguments:

`load_helpers` Whether to load test helpers

`load_package` The package loading strategy

Method `execute()`**:** Execute the test strategy

Usage:

```
FullTestStrategy$execute(path, plan, reporter)
```

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `muttest_plan()`.

`reporter` The reporter to use for test results

Returns: The test results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FullTestStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FileTestStrategy`, `TestStrategy`, `default_test_strategy()`

`index_decrement` *Decrement subscript indices*

Description

Replaces every simple subscript index in `x[i]` or `x[[i]]` with `x[i - 1L]` / `x[[i - 1L]]`. Targets identifier and numeric literal indices; complex expressions (e.g. `x[a + b]`) are left untouched.

Usage

```
index_decrement()
```

Value

A Mutator object.

Examples

```
index_decrement()
```

`index_increment` *Increment subscript indices*

Description

Replaces every simple subscript index in `x[i]` or `x[[i]]` with `x[i + 1L]` / `x[[i + 1L]]`. Targets identifier and numeric literal indices; complex expressions (e.g. `x[a + b]`) are left untouched.

Usage

```
index_increment()
```

Details

Catches off-by-one errors where tests never verify the exact element retrieved from a vector or list.

Value

A Mutator object.

Examples

```
index_increment()
```

`index_mutations` *Index mutation mutators*

Description

Returns a ready-made list of `index_increment()` and `index_decrement()` mutators that shift every simple subscript index by 1.

Usage

```
index_mutations()
```

Details

Use on any file that extracts elements from vectors or lists by position. Surviving mutants reveal off-by-one errors where tests only verify that *something* was extracted, not *which* element — asserting the exact value of the extracted element kills them.

Value

A list of mutators.

See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
index_mutations()

## Not run:
plan <- muttest_plan(
  source_files = "R/selectors.R",
  mutators = index_mutations()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

`JSONMutationReporter` *JSON Reporter for Mutation Testing*

Description

A quiet reporter that writes a machine-readable JSON artifact. It prints nothing while running; on completion it writes one JSON file describing every mutant, keyed by source file. The JSON is the data contract consumed by dashboards, CI job summaries, and the HTML report (see `report()`).

Combine it with ProgressMutationReporter via MultiReporter to get a live console display and a JSON file from the same run.

Super class

`muttest::MutationReporter` -> `JSONMutationReporter`

Public fields

`path` Path of the JSON file to write.

`mutants_by_file` Per-file lists of mutant records.

`sources` Per-file original source lines.

Methods

Public methods:

- `JSONMutationReporter$new()`
- `JSONMutationReporter$start_reporter()`
- `JSONMutationReporter$add_result()`
- `JSONMutationReporter$end_reporter()`
- `JSONMutationReporter$clone()`

Method `new()`**:** Initialize a new JSON reporter

Usage:

```
JSONMutationReporter$new(path = "muttest.json", ...)
```

Arguments:

`path` Path of the JSON file to write (default: `"muttest.json"`).

`...` Passed to the MutationReporter constructor.

Method `start_reporter()`**:** Start reporter

Usage:

```
JSONMutationReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

Method `add_result()`**:** Add a mutation test result

Usage:

```
JSONMutationReporter$add_result(
  plan,
  killed,
  survived,
  no_coverage,
  errors,
  error = NULL,
  original_code = NULL,
  mutated_code = NULL
)
```

Arguments:

`plan` Current testing plan. See `muttest_plan()`.

`killed` Whether the mutation was killed by tests

`survived` Number of survived mutations

`no_coverage` Number of mutants with no test coverage

`errors` Number of errors encountered

`error` Optional error condition from a failed run

`original_code` Original source lines before mutation

`mutated_code` Mutated source lines

Method `end_reporter()`**:** End reporter, then write the JSON file

Usage:

```
JSONMutationReporter$end_reporter()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
JSONMutationReporter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `MultiReporter`, `MutationReporter`, `ProgressMutationReporter`, `default_reporter()`

`logical_operators` *Logical operator mutators*

Description

Returns a ready-made list of `operator()` mutators covering short-circuit (`&&`/`||`) and vectorised (`&`/`|`) logical operator swaps.

Usage

```
logical_operators()
```

Details

Use on any file with compound conditions (`if (a && b)`). A surviving mutant from this preset typically means test inputs are symmetric — both flags `TRUE` or both `FALSE`. Adding a test with one flag `TRUE` and the other `FALSE` exposes the difference between `&&` and `||` and kills the mutant.

Value

A list of `operator()` mutators.

See Also

`vignette("mutators", package = "muttest")` for the full operator table and a worked example showing the symmetric-input pattern.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
logical_operators()

## Not run:
plan <- muttest_plan(
  source_files = "R/access.R",
  mutators = logical_operators()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

`MultiReporter` *Combine Several Mutation Reporters*

Description

Fans out every reporter event to a list of child reporters, so a single `muttest()` run can drive more than one reporter at once – for example a live ProgressMutationReporter alongside a JSONMutationReporter that writes a machine-readable artifact.

Super class

`muttest::MutationReporter` -> `MultiReporter`

Public fields

`reporters` List of child reporters.

Methods

Public methods:

- MultiReporter$new()
- MultiReporter$start_reporter()
- MultiReporter$start_file()
- MultiReporter$start_mutator()
- MultiReporter$add_result()
- MultiReporter$update()
- MultiReporter$end_mutator()
- MultiReporter$end_file()
- MultiReporter$end_reporter()
- MultiReporter$get_score()
- MultiReporter$print()
- MultiReporter$clone()

Method `new()`**:** Initialize a combined reporter

Usage:

```
MultiReporter$new(...)
```

Arguments:

`...` One or more MutationReporter objects.

Method `start_reporter()`**:** Start reporter

Usage:

```
MultiReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

Method `start_file()`**:** Start testing a file

Usage:

```
MultiReporter$start_file(filename)
```

Arguments:

`filename` Path to the file being mutated

Method `start_mutator()`**:** Start testing with a specific mutator

Usage:

```
MultiReporter$start_mutator(mutator)
```

Arguments:

`mutator` The mutator being applied

Method `add_result()`**:** Add a mutation test result

Usage:

```
MultiReporter$add_result(...)
```

Arguments:

`...` Arguments forwarded to each child's `add_result()`.

Method `update()`**:** Update status

Usage:

```
MultiReporter$update(force = FALSE)
```

Arguments:

`force` Passed to each child

Method `end_mutator()`**:** End testing with current mutator

Usage:

```
MultiReporter$end_mutator()
```

Method `end_file()`**:** End testing current file

Usage:

```
MultiReporter$end_file()
```

Method `end_reporter()`**:** End reporter

Usage:

```
MultiReporter$end_reporter()
```

Method `get_score()`**:** Get the score (from the first child)

Usage:

```
MultiReporter$get_score()
```

Method `print()`**:** Print each child that supports printing

Usage:

```
MultiReporter$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MultiReporter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `JSONMutationReporter`, `MutationReporter`, `ProgressMutationReporter`, `default_reporter()`

`MutationReporter` *Reporter for Mutation Testing*

Description

The job of a mutation reporter is to aggregate and display the results of mutation tests. It tracks each mutation attempt, reporting on whether the tests killed the mutation or the mutation survived.

Public fields

`test_reporter` Reporter to use for the testthat::test_dir function

`out` Output destination for reporter messages

`width` Width of the console in characters

`unicode` Whether Unicode output is supported

`crayon` Whether colored output is supported

`rstudio` Whether running in RStudio

`hyperlinks` Whether terminal hyperlinks are supported

`current_file` Path of the file currently being mutated

`current_mutator` Mutator currently being applied

`plan` Complete mutation plan for the test run

`results` List of mutation test results, indexed by file path

`current_score` Current score of the mutation tests

`error_messages` List of error messages from failed mutant runs

Methods

Public methods:

- `MutationReporter$new()`
- `MutationReporter$start_reporter()`
- `MutationReporter$start_file()`
- `MutationReporter$start_mutator()`
- `MutationReporter$add_result()`
- `MutationReporter$update()`
- `MutationReporter$end_mutator()`
- `MutationReporter$end_file()`
- `MutationReporter$end_reporter()`
- `MutationReporter$get_score()`
- `MutationReporter$cat_line()`
- `MutationReporter$rule()`
- `MutationReporter$clone()`

Method `new()`**:** Initialize a new reporter

Usage:

```
MutationReporter$new(test_reporter = "silent", file = stdout())
```

Arguments:

`test_reporter` Reporter to use for the testthat::test_dir function

`file` Output destination (default: stdout)

Method `start_reporter()`**:** Start reporter

Usage:

```
MutationReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

`temp_dir` Path to the temporary directory for testing

Method `start_file()`**:** Start testing a file

Usage:

```
MutationReporter$start_file(filename)
```

Arguments:

`filename` Path to the file being mutated

Method `start_mutator()`**:** Start testing with a specific mutator

Usage:

```
MutationReporter$start_mutator(mutator)
```

Arguments:

`mutator` The mutator being applied

Method `add_result()`**:** Add a mutation test result

Usage:

```
MutationReporter$add_result(
  plan,
  killed,
  survived,
  no_coverage,
  errors,
  error = NULL,
  original_code = NULL,
  mutated_code = NULL
)
```

Arguments:

`plan` Current testing plan. See `muttest_plan()`.

`killed` Whether the mutation was killed by tests

`survived` Number of survived mutations

`no_coverage` Number of mutants with no test coverage

`errors` Number of errors encountered
`error` Optional error condition from a failed run
`original_code` Original source lines before mutation
`mutated_code` Mutated source lines

Method `update()`**:** Update status (no-op in base class)

Usage:

```
MutationReporter$update(force = FALSE)
```

Arguments:

`force` Ignored

Method `end_mutator()`**:** End testing with current mutator

Usage:

```
MutationReporter$end_mutator()
```

Method `end_file()`**:** End testing current file

Usage:

```
MutationReporter$end_file()
```

Method `end_reporter()`**:** End reporter and show summary

Usage:

```
MutationReporter$end_reporter()
```

Method `get_score()`**:** Get the current score

Usage:

```
MutationReporter$get_score()
```

Method `cat_line()`**:** Print a message to the output

Usage:

```
MutationReporter$cat_line(...)
```

Arguments:

`...` Message to print

Method `rule()`**:** Print a message to the output with a rule

Usage:

```
MutationReporter$rule(...)
```

Arguments:

`...` Message to print

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MutationReporter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `JSONMutationReporter`, `MultiReporter`, `ProgressMutationReporter`, `default_reporter()`

`Mutator` *Mutator*

Description

Mutator

Mutator

Details

Represents a single code mutation — a pattern to find and a replacement to apply. Every mutator function (`operator()`, `boolean_literal()`, etc.) returns an instance of this class.

Public fields

`from` The token or operator to replace.

`to` The replacement token or operator.

`query` Tree-sitter query used to locate candidate nodes.

`match_fn` Optional `function(node_text)` returning `logical`; overrides the default `node_text == from` equality check.

`replacement_fn` Optional `function(node_text)` returning a string; overrides the static `to` value as the replacement text.

`mutate_fn` Optional `function(code)` that fully replaces the default mutation logic when set.

Methods

Public methods:

- `Mutator$new()`
- `Mutator$mutate()`
- `Mutator$print()`
- `Mutator$clone()`

Method `new()`**:** Create a new `Mutator`.

Usage:

```
Mutator$new(
  from,
  to,
  query,
  match_fn = NULL,
  replacement_fn = NULL,
  mutate_fn = NULL
)
```

Arguments:

`from` Token to replace.

`to` Replacement token.

`query` Tree-sitter query string.

`match_fn` Optional custom match function.

`replacement_fn` Optional custom replacement function.

`mutate_fn` Optional `function(code)` that fully overrides the default mutation logic.

Method `mutate()`**:** Apply this mutator to a character vector of source lines.

Usage:

```
Mutator$mutate(code)
```

Arguments:

`code` Character vector of source lines.

Returns: A list of mutation records (one per match), each a list with `code` (mutated source lines), `location` (1-based `start`/`end` line/column), and `replacement` (the inserted text), or `NULL` if the pattern was not found.

Method `print()`**:** Print a short summary of the mutator.

Usage:

```
Mutator$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Mutator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`muttest` *Run a mutation test*

Description

Run a mutation test

Usage

```
muttest(
  plan,
  path = "tests/testthat",
  reporter = default_reporter(),
  test_strategy = default_test_strategy(),
  copy_strategy = default_copy_strategy(),
  workers = 1,
  timeout = 600 * 1000
)
```

Arguments

plan	A mutation testing plan. See muttest_plan().
path	Path to the test directory.
reporter	Reporter to use for mutation testing results. See MutationReporter.
test_strategy	Strategy for running tests. See TestStrategy. The purpose of test strategy is to control how tests are executed. We can run all tests for each mutant, or only tests that are relevant to the mutant.
copy_strategy	Strategy for copying the project. See CopyStrategy. This strategy controls which files are copied to the temporary directory, where the tests are run.
workers	Number of parallel workers. When greater than 1, mutants are tested concurrently using mirai daemons. Defaults to 1 (sequential).
timeout	Per-mutant timeout in milliseconds. If a mutant's test run exceeds this limit the daemon is interrupted and the result is recorded as an error. Use Inf to disable. Defaults to 600000 (10 minutes).

Value

An object of class muttest_result containing the overall mutation score.

---

muttest_plan *Create a plan for mutation testing*

Description

Each mutant requires rerunning the tests. For large project it might be not feasible to test all mutants in one go. This function allows you to create a plan for selected source files and mutators.

Usage

```
muttest_plan(mutators, source_files = fs::dir_ls("R", regexp = ".[rR]$"))
```

Arguments

mutators	A list of mutators to use. See operator().
source_files	A vector of file paths to the source files.

Details

The plan is in a data frame format, where each row represents a mutant.

You can subset the plan before passing it to the muttest() function.

Value

A data frame with the test plan. The data frame has the following columns:

- `filename`: The name of the source file.
- `original_code`: The original code of the source file.
- `mutated_code`: The mutated code of the source file.
- `mutator`: The mutator that was applied.
- `mutation`: A list with the mutant's `location` (1-based `start`/`end` line/column) and `replacement` text.

`na_literal` *Mutate an NA or NULL literal*

Description

Replaces `NA`, `NULL`, or a typed NA constant (`NA_real_`, `NA_integer_`, `NA_complex_`, `NA_character_`) with another value.

Usage

```
na_literal(from, to)
```

Arguments

`from`	The literal to replace. One of `"NA"`, `"NULL"`, `"NA_real_"`, `"NA_integer_"`, `"NA_complex_"`, `"NA_character_"`.
`to`	The replacement literal.

Details

In tree-sitter-r, `NA` and all typed NA variants share the same `na` node type, while `NULL` has its own `null` node type. `match_fn` distinguishes between them by comparing the literal text.

Value

A Mutator object.

Examples

```
na_literal("NULL", "NA")
na_literal("NA", "NULL")
na_literal("NA", "NA_real_")
na_literal("NA_real_", "NA")
```

na_literals *NA and NULL literal mutators*

Description

Returns a ready-made list of `na_literal()` mutators covering common swaps between `NA`, `NULL`, and the typed NA variants (`NA_real_`, `NA_integer_`, `NA_character_`).

Usage

```
na_literals()
```

Details

Use on any file that handles missing values or nullable results. A surviving mutant typically means tests never distinguish `NA` from `NULL`, or never check which typed NA is returned — adding `expect_true(is.na(x))` and `expect_equal(class(x), "numeric")` style assertions kills it.

Value

A list of `na_literal()` mutators.

See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
na_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/missing.R",
  mutators = na_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

negate_condition *Negate the condition of if/while statements*

Description

Wraps the condition expression of each matching statement in `!(...)`. For example, `if (x > 0)` becomes `if (!(x > 0))`.

Usage

```
negate_condition(statements = c("if", "while"))
```

Arguments

`statements`	Character vector of statement types to target. Must be a subset of `c("if", "while")`. Defaults to both.

Value

A Mutator object.

Examples

```
negate_condition()
negate_condition(statements = "if")
```

numeric_decrement *Decrement numeric literals*

Description

Replaces every numeric literal `n` with `n - by`. Handles both integer (e.g. `5L`) and floating-point (e.g. `3.14`) literals.

Usage

```
numeric_decrement(by = 1)
```

Arguments

`by`	The amount to subtract. Defaults to `1`.

Value

A Mutator object.

Examples

```
numeric_decrement()
numeric_decrement(by = 2)
```

`numeric_increment` *Increment numeric literals*

Description

Replaces every numeric literal `n` with `n + by`. Handles both integer (e.g. `5L`) and floating-point (e.g. `3.14`) literals.

Usage

```
numeric_increment(by = 1)
```

Arguments

`by`	The amount to add. Defaults to `1`.

Value

A Mutator object.

Examples

```
numeric_increment()
numeric_increment(by = 2)
```

`numeric_literals` *Numeric literal mutators*

Description

Returns a ready-made list of `numeric_increment()` and `numeric_decrement()` mutators that shift every numeric literal by 1.

Usage

```
numeric_literals()
```

Details

Use on any file with numeric constants used as thresholds, counts, or offsets. A surviving mutant means tests never verify the exact value of the constant — asserting the precise numeric result rather than a property (e.g. sign) kills it.

Value

A list of mutators.

See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
numeric_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/thresholds.R",
  mutators = numeric_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

---

operator *Mutate a binary operator*

Description

Produces one mutant per occurrence of `from` in the source file, replacing it with `to`. A surviving mutant means your tests cannot distinguish the original operator from the replacement — pointing at the missing assertion or input value.

Usage

```
operator(from, to)
```

Arguments

`from`	The operator to replace (e.g. `"+"`, `"=="`, `">"`).
`to`	The replacement operator.

Details

Use this when you need a specific swap not covered by the preset collections (`arithmetic_operators()`, `comparison_operators()`, `logical_operators()`).

Value

A Mutator object.

See Also

comparison_operators(), arithmetic_operators(), logical_operators() for ready-made preset lists.

vignette("mutators", package = "muttest") for the full operator reference with examples of what each preset catches.

vignette("interpreting-results", package = "muttest") to learn how to read surviving mutants and strengthen the tests they expose.

Examples

```
operator("+", "-")
operator("==", "!=")
operator(">", ">=")  # probe the strict vs. non-strict boundary
```

PackageCopyStrategy *Package copy strategy*

Description

It copies all files and directories from the original directory to a temporary directory.

When symlink = TRUE, only the top-level directory holding the mutated file (e.g. R/) is copied; every other top-level directory and root file is symlinked to the original. For N mutants this avoids N full copies, which matters when the project carries large inst//data/ directories. Tests only read the symlinked files, so this is safe as long as the test suite does not write into the package's own directories (writes through a symlink would modify the original).

Super class

muttest::CopyStrategy -> PackageCopyStrategy

Methods

Public methods:

- PackageCopyStrategy$new()
- PackageCopyStrategy$execute()
- PackageCopyStrategy$clone()

Method new(): Create a package copy strategy

Usage:

```
PackageCopyStrategy$new(symlink = FALSE)
```

Arguments:

symlink If TRUE, symlink unchanged files instead of copying them.

Method execute(): Copy project files, excluding hidden and temp directories

Usage:

```
PackageCopyStrategy$execute(original_dir, plan)
```

Arguments:

`original_dir` The original directory to copy from

`plan` The current test plan

Returns: The path to the temporary directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PackageCopyStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other CopyStrategy: `CopyStrategy`, `default_copy_strategy()`

`ProgressMutationReporter`

Progress Reporter for Mutation Testing

Description

A reporter that displays a progress indicator for mutation tests. It provides real-time feedback on which mutants are being tested and whether they were killed by tests.

Super class

`muttest::MutationReporter` -> `ProgressMutationReporter`

Public fields

`start_time` Time when testing started (for duration calculation)

`min_time` Minimum test duration to display timing information

`col_config` List of column configuration for report formatting

`survived_detail` Controls how survived mutants are reported (summary, none)

`survived_mutants` List to store details of survived mutants for summary reporting

Methods

Public methods:

- ProgressMutationReporter$format_column()
- ProgressMutationReporter$fmt_h()
- ProgressMutationReporter$fmt_r()
- ProgressMutationReporter$new()
- ProgressMutationReporter$start_reporter()
- ProgressMutationReporter$add_result()
- ProgressMutationReporter$update()
- ProgressMutationReporter$end_file()
- ProgressMutationReporter$end_reporter()
- ProgressMutationReporter$print()
- ProgressMutationReporter$clone()

Method `format_column()`**:** Format a column with specified padding and width

Usage:

```
ProgressMutationReporter$format_column(text, col_name, colorize = NULL)
```

Arguments:

`text` Text to format

`col_name` Column name to use configuration from

`colorize` Optional function to color the text

Method `fmt_h()`**:** Format the header of the report

Usage:

```
ProgressMutationReporter$fmt_h()
```

Method `fmt_r()`**:** Format a row of the report

Usage:

```
ProgressMutationReporter$fmt_r(status, k, s, n, e, t, score, mutator, file)
```

Arguments:

`status` Status symbol (e.g., tick or cross)

`k` Number of killed mutations

`s` Number of survived mutations

`n` Number of mutations with no test coverage

`e` Number of errors

`t` Total number of mutations

`score` Score percentage

`mutator` The mutator used

`file` The file being tested

Returns: Formatted row string

Method `new()`**:** Initialize a new progress reporter

Usage:

```
ProgressMutationReporter$new(
  test_reporter = "silent",
  min_time = 1,
  file = stdout(),
  survived_detail = c("summary", "none")
)
```

Arguments:

`test_reporter` Reporter to use for testthat::test_dir

`min_time` Minimum time to show elapsed time (default: 1s)

`file` Output destination (default: stdout)

`survived_detail` Controls how survived mutants are reported. One of `"summary"` (default) or `"none"`.

Method `start_reporter()`**:** Start reporter

Usage:

```
ProgressMutationReporter$start_reporter(plan = NULL)
```

Arguments:

`plan` The complete mutation plan

Method `add_result()`**:** Add a mutation test result

Usage:

```
ProgressMutationReporter$add_result(
  plan,
  killed,
  survived,
  no_coverage,
  errors,
  error = NULL,
  original_code = NULL,
  mutated_code = NULL
)
```

Arguments:

`plan` Current testing plan. See `muttest_plan()`.

`killed` Whether the mutation was killed by tests

`survived` Number of survived mutations

`no_coverage` Number of mutants with no test coverage

`errors` Number of errors encountered

`error` Optional error condition from a failed run

`original_code` Original source lines before mutation

`mutated_code` Mutated source lines

Method `update()`**:** Update status spinner (for long-running operations)

Usage:

```
ProgressMutationReporter$update(force = FALSE)
```

Arguments:

`force` Force update even if interval hasn't elapsed

Method `end_file()`**:** End testing current file

Usage:

```
ProgressMutationReporter$end_file()
```

Method `end_reporter()`**:** End reporter with detailed summary

Usage:

```
ProgressMutationReporter$end_reporter()
```

Method `print()`**:** Print the mutation test result with survived diffs

Usage:

```
ProgressMutationReporter$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ProgressMutationReporter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other MutationReporter: `JSONMutationReporter`, `MultiReporter`, `MutationReporter`, `default_reporter()`

`remove_condition_negation` *Remove negation from the condition of if/while statements*

Description

The inverse of `negate_condition()`. Strips the leading `!` from any already-negated condition, so `if (!done)` becomes `if (done)` and `while (!ready)` becomes `while (ready)`.

Usage

```
remove_condition_negation(statements = c("if", "while"))
```

Arguments

`statements`	Character vector of statement types to target. Must be a subset of `c("if", "while")`. Defaults to both.

Details

Unlike `remove_negation()`, this mutator is scoped exclusively to conditions, leaving negations in other positions (assignments, return values, etc.) untouched.

Value

A Mutator object.

Examples

```
remove_condition_negation()
remove_condition_negation(statements = "while")
```

---

remove_negation *Remove logical negation*

Description

Removes the `!` unary operator from an expression. For example, `!is.na(x)` becomes `is.na(x)` and `!(a > b)` becomes `(a > b)`.

Usage

```
remove_negation()
```

Value

A Mutator object.

Examples

```
remove_negation()
```

---

replace_return_value *Replace the value in explicit return() calls*

Description

Replaces the argument of every `return(expr)` with a fixed value (default `"NULL"`). Tests that only check that a function returns *something* without asserting the value will not kill these mutants.

Usage

```
replace_return_value(replacement = "NULL")
```

Arguments

`replacement`	Raw R source text to substitute as the return value. Defaults to `"NULL"`. Examples: `"NA"` inserts the missing value `NA`; `'"NULL"'` (inner quotes) inserts the string `"NULL"`; `'"NA"'` inserts the string `"NA"` rather than the missing value.

Details

Only explicit `return()` calls are targeted. Implicit returns (the last expression of a function body) are not affected.

Value

A Mutator object.

Examples

```
replace_return_value()
replace_return_value("NA")
```

`report` *Build a source-annotated HTML report from a muttest JSON file*

Description

Reads a JSON file produced by JSONMutationReporter (the mutation-testing-elements schema) and renders a self-contained HTML report: overall score, per-file breakdown, and every mutant overlaid on its source line. Mutated lines expand to show each mutation as a diff. A small inline script adds status filtering (the report opens focused on survived mutants) and n/p keyboard navigation between mutated lines.

Usage

```
report(json = "muttest.json", output = sub("\\.json$", ".html", json))
```

Arguments

`json`	Path to the JSON file written by JSONMutationReporter.
`output`	Path of the HTML file to write. Defaults to the JSON path with an `.html` extension.

Details

The report is a single static file. Syntax highlighting is loaded from a CDN (shiki) when online and is skipped gracefully offline.

Value

The `output` path, invisibly.

string_empty *Mutate non-empty string literals to the empty string*

Description

Replaces any non-empty string literal in the source code with `""`. The empty string itself is not mutated (use `string_fill()` for that).

Usage

```
string_empty()
```

Value

A Mutator object.

Examples

```
string_empty()
```

string_fill *Mutate the empty string literal to a placeholder string*

Description

Replaces `""` with a fill string (default `"mutant"`) so that code paths that depend on an empty string can be detected.

Usage

```
string_fill(fill = "mutant")
```

Arguments

`fill`	The replacement string. Defaults to `"mutant"`. Override when the codebase already contains `"mutant"` as a meaningful value.

Value

A Mutator object.

Examples

```
string_fill()
string_fill(fill = "PLACEHOLDER")
```

`string_literals` *String literal mutators*

Description

Returns a ready-made list of `string_empty()` and `string_fill()` mutators covering both directions: collapsing non-empty strings to `""` and filling empty strings with a placeholder.

Usage

```
string_literals()
```

Details

Use on any file where string values are passed to downstream logic or returned to callers. Surviving mutants reveal tests that only check *type* or *length* — asserting the exact string content kills them.

Value

A list of mutators.

See Also

`vignette("mutators", package = "muttest")` for the full mutator table.

`vignette("interpreting-results", package = "muttest")` for how to diagnose survivors and fix the underlying test weakness.

Examples

```
string_literals()

## Not run:
plan <- muttest_plan(
  source_files = "R/labels.R",
  mutators = string_literals()
)
muttest(plan, "tests/testthat")

## End(Not run)
```

`TestStrategy` *TestStrategy interface*

Description

Extend this class to implement a custom test strategy.

Methods

Public methods:

- `TestStrategy$execute()`
- `TestStrategy$clone()`

Method `execute()`**:** Execute the test strategy

Usage:

```
TestStrategy$execute(path, plan, reporter)
```

Arguments:

`path` The path to the test directory

`plan` The current mutation plan. See `muttest_plan()`.

`reporter` The reporter to use for test results

Returns: The test result

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TestStrategy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other TestStrategy: `FileTestStrategy`, `FullTestStrategy`, `default_test_strategy()`

Index