

Package: tinyexpect (via r-universe)

September 18, 2026

Title Additional Expectation Predicates for the 'tinytest' Testing Framework

Version 0.0.0.9010

Description Ports of 'testthat' expectation predicates to the 'tinytest' testing framework. These ports allow easier transitioning from 'testthat' to 'tinytest' as well as provide more explicit expectation predicates for 'tinytest' users.

License MIT + file LICENSE

URL <https://mojaveazure.github.io/tinyexpect>,
<https://github.com/mojaveazure/tinyexpect>

BugReports <https://github.com/mojaveazure/tinyexpect/issues>

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5), tinytest (>= 1.4.1)

Imports rlang (>= 1.1.3), utils

Suggests Matrix, methods, withr

Config/Needs/xrefs testthat

Repository <https://mojaveazure.r-universe.dev>

Date/Publication 2025-11-18 01:43:30 UTC

RemoteUrl <https://github.com/mojaveazure/tinyexpect>

RemoteRef HEAD

RemoteSha 8f780b40ee385d473ecec9905239b0d2c06f39d2

Contents

expect_comparisons	2
expect_condition	3
expect_contains	4
expect_in	5

expect_invisible	6
expect_mapequal	7
expect_named	8
expect_no_condition	9
expect_s3_class	10
expect_s4_class	11
expect_setequal	12
expect_type	13
expect_visible	13
skip	14
skip_if_not_installed	15
skip_on_bioc	16
skip_on_ci	17
skip_on_covr	18
skip_on_cran	19
skip_on_os	20

Index	22
--------------	-----------

expect_comparisons	<i>Expect a Comparison to Hold</i>
--------------------	------------------------------------

Description

Expectation predicate for testing based on [comparisons](#)

Usage

```
expect_lt(current, target, ..., info = NA_character_)
```

```
expect_lte(current, target, ..., info = NA_character_)
```

```
expect_gt(current, target, ..., info = NA_character_)
```

```
expect_gte(current, target, ..., info = NA_character_)
```

Arguments

current	[R object or expression] Object or expression to be tested
target	[R object or expression] Expected outcome
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalents: [expect_lt\(\)](#), [expect_lte\(\)](#), [expect_gt\(\)](#), and [expect_gte\(\)](#)

Examples

```
# Expect strictly less than
expect_lt(1L, 3L) # Pass
expect_lt(3L, 1L) # Fail
expect_lt(1L, 1L) # Fail

# Expect less than or equal to
expect_lte(1L, 3L) # Pass
expect_lte(3L, 1L) # Fail
expect_lte(1L, 1L) # Pass

# Expect strictly greater than
expect_gt(1L, 3L) # Fail
expect_gt(3L, 1L) # Pass
expect_gt(1L, 1L) # Fail

# Expect greater than or equal to
expect_gte(1L, 3L) # Fail
expect_gte(3L, 1L) # Pass
expect_gte(1L, 1L) # Pass
```

expect_condition	<i>Expect a Condition to be Thrown</i>
------------------	--

Description

Expectation predicate for testing that a given function call throws a [condition](#)

Usage

```
expect_condition(
  current,
  pattern = NULL,
  class = NULL,
  ...,
  info = NA_character_
)
```

Arguments

current	[R object or expression] Object or expression to be tested
pattern	[character(1L)] A regular expression to match the condition message; any condition thrown that does not match pattern will trigger a failure. Must be a single character string

class	[character] Optional vector of classes from which the condition should inherit; any conditions thrown that do not inherit from class will trigger a failure
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

[testthat](#) equivalent: [testthat::expect_condition\(\)](#)

Other condition expectation predicates: [expect_no_condition\(\)](#)

Examples

```
expect_condition(message("test message")) # Pass
expect_condition(warning("test warning")) # Pass
expect_condition(stop("test error")) # Pass

expect_condition("tomato") # Fail
expect_condition(message("test message"), class = "specialMessage") # Fail
expect_condition(message("test message"), pattern = "tomato") # Fail
```

expect_contains	<i>Expect a Superset</i>
-----------------	--------------------------

Description

Expectation predicate for testing that a given vector contains all values present in another vector

Usage

```
expect_contains(current, target, ..., info = NA_character_)
```

Arguments

current	[R vector] A vector to be tested
target	[R vector] A vector to be compared against
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_contains()`

Other in-condition expectation predicates: `expect_in()`, `expect_mapequal()`, `expect_setequal()`

Examples

```
expect_contains(letters, target = rev(letters)) # Pass
expect_contains(letters, target = "b") # Pass
expect_contains("b", target = letters) # Fail
```

expect_in	<i>Expect a Subset</i>
-----------	------------------------

Description

Expectation predicate for testing that a vector is fully contained within another vector

Usage

```
expect_in(current, target, ..., info = NA_character_)
```

Arguments

current	[R vector] A vector to be tested
target	[R vector] A vector to be compared against
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_in()`

Other in-condition expectation predicates: `expect_contains()`, `expect_mapequal()`, `expect_setequal()`

Examples

```
expect_in(letters, target = rev(letters)) # Pass
expect_in(letters, target = "b") # Fail
expect_in("b", target = letters) # Pass
```

expect_invisible	<i>Expect Invisible</i>
------------------	-------------------------

Description

Expect Invisible

Usage

```
expect_invisible(current, ..., info = NA_character_)
```

Arguments

current	[R object or expression] Object or expression to be tested
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A `tinytest` object. A `tinytest` object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_invisible()`

`withVisible()`, `invisible()`

Other visibility expectation predicate: `expect_visible()`

Examples

```
f <- function() invisible()
expect_invisible(f()) # Pass
```

```
g <- function() NULL
expect_invisible(g()) # Fail
```

expect_mapequal	<i>Expect Map Equality</i>
-----------------	----------------------------

Description

Expectation predicate for testing that the named [sets](#) of two vectors are equal

Usage

```
expect_mapequal(  
  current,  
  target,  
  ...,  
  info = NA_character_,  
  tolerance = sqrt(x = .Machine$double.eps)  
)
```

Arguments

current	[R vector] A vector to be tested
target	[R vector] A vector to be compared against
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value
tolerance	[numeric] Test equality to machine rounding; passed to all.equal()

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: [testthat::expect_mapequal\(\)](#)

Other in-condition expectation predicates: [expect_contains\(\)](#), [expect_in\(\)](#), [expect_setequal\(\)](#)

Examples

```
(x <- `names<-`(letters, letters))  
expect_mapequal(x, target = rev(x)) # Pass  
expect_mapequal(letters, target = c(b = "b")) # Fail  
expect_mapequal(c(b = "b"), target = x) # Fail
```

expect_named	<i>Expect Names</i>
--------------	---------------------

Description

Expectation predicate for testing the [names](#) of an object

Usage

```
expect_named(  
  current,  
  target,  
  ...,  
  ignore.order = FALSE,  
  ignore.case = FALSE,  
  info = NA_character_  
)
```

Arguments

current	[R object or expression] Object or expression to be tested
target	[missing, character, or NULL] One of the following: • Left missing to test that current has names • A character vector giving the expected names of current • NULL to test that current is unnamed
...	Ignored
ignore.order	[logical(1L)] Ignore the order of names(current) with respect to target
ignore.case	[logical(1L)] Ignore upper- vs lower-case discrepancies between names(current) and target
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: [testthat::expect_named\(\)](#)

Examples

```
(x <- c(a = 1L, b = 2L, c = 3L))
(y <- c(4L, 5L, 6L))

expect_named(x) # Pass
expect_named(y) # Fail

expect_named(x, target = NULL) # Fail
expect_named(y, target = NULL) # Pass

expect_named(x, target = c("a", "b", "c")) # Pass
expect_named(y, target = c("a", "b", "c")) # Fail

expect_named(x, target = c("b", "a", "c")) # Fail
expect_named(x, target = c("b", "a", "c"), ignore.order = TRUE) # Pass

expect_named(x, target = c("A", "B", "C")) # Fail
expect_named(x, target = c("A", "B", "C"), ignore.case = TRUE) # Pass
```

expect_no_condition	<i>Expect No Conditions to be Thrown</i>
---------------------	--

Description

Expectation predicate for testing that a given function does not raise a [condition](#)

Usage

```
expect_no_condition(
  current,
  pattern = NULL,
  class = NULL,
  ...,
  info = NA_character_
)
```

Arguments

current	[R object or expression] Object or expression to be tested
pattern	[character(1L)] A regular expression to match the condition message; any conditions thrown with a message matching pattern will trigger a failure while all other conditions thrown will trigger a success. Must be a single character string
class	[character] Optional vector of condition classes that should not be thrown; conditions thrown that inherit from class will trigger a failure while all other conditions thrown will trigger a success. If NULL, any condition thrown will trigger a failure

...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A `tinytest` object. A `tinytest` object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_no_condition()`

Other condition expectation predicates: `expect_condition()`

Examples

```
expect_no_condition("tomato") # Pass

expect_no_condition(message("test")) # Fail

expect_no_condition(message("test"), pattern = "test") # Fail
expect_no_condition(message("test"), pattern = "tomato") # Pass

expect_no_condition(message("test"), class = "message") # Fail
expect_no_condition(message("test"), class = "error") # Pass

expect_no_condition(message("test"), pattern = "test", class = "message") # Fail
expect_no_condition(message("test"), pattern = "tomato", class = "message") # Pass
expect_no_condition(message("test"), pattern = "test", class = "error") # Pass
expect_no_condition(message("test"), pattern = "tomato", class = "error") # Pass
```

expect_s3_class	<i>Expect S3 Object</i>
-----------------	-------------------------

Description

Check that current is an S3 object that [inherits](#) from class

Usage

```
expect_s3_class(current, class, ..., info = NA_character_, exact = FALSE)
```

Arguments

current	[R object or expression] Object or expression to be tested
class	[character] Expected class of current; may optionally pass NA to assert that current is not an object
...	Ignored

info	[character(1L)] Optional user-defined message; must be a single string value
exact	[logical(1L)] If TRUE, check that <code>class(current)</code> is identical to <code>class</code> ; otherwise, check that current inherits class

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

[testthat](#) equivalent: [testthat::expect_s3_class\(\)](#)

Other typing expectation predicates: [expect_s4_class\(\)](#), [expect_type\(\)](#)

Examples

```
expect_s3_class(data.frame(), "data.frame") # Pass
expect_s3_class(1L, "integer") # Fail
expect_s3_class(1L, NA) # Pass
```

```
expect_s3_class(Matrix::Matrix(), "Matrix") # Fail
expect_s3_class(Matrix::Matrix(), NA) # Pass
```

expect_s4_class	<i>Expect S4 Object</i>
-----------------	-------------------------

Description

Check that current is an S4 object that is of class class

Usage

```
expect_s4_class(current, class, ..., info = NA_character_)
```

Arguments

current	[R object or expression] Object or expression to be tested
class	[character] Expected class of current; may optionally pass NA to assert that current is not an object
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_s4_class()`

Other typing expectation predicates: `expect_s3_class()`, `expect_type()`

Examples

```
expect_s4_class(Matrix::Matrix(), "Matrix") # Pass
```

```
expect_s4_class(data.frame(), "data.frame") # Fail
```

```
expect_s4_class(data.frame(), NA) # Pass
```

expect_setequal	<i>Expect Set Equality</i>
-----------------	----------------------------

Description

Expectation predicate for testing that the [sets](#) of two vectors are equal

Usage

```
expect_setequal(current, target, ..., info = NA_character_)
```

Arguments

current	[R vector] A vector to be tested
target	[R vector] A vector to be compared against
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A [tinytest](#) object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_setequal()`

Other in-condition expectation predicates: `expect_contains()`, `expect_in()`, `expect_mapequal()`

Examples

```
expect_setequal(letters, target = rev(letters)) # Pass
```

```
expect_setequal(letters, target = "b") # Fail
```

```
expect_setequal("b", target = letters) # Fail
```

expect_type	<i>Expect Object Type</i>
-------------	---------------------------

Description

Check that `typeof(current)` is type

Usage

```
expect_type(current, type, ..., info = NA_character_)
```

Arguments

current	[R object or expression] Object or expression to be tested
type	[character(1L)] Expected base type (as given by <code>typeof()</code>)
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A `tinytest` object. A `tinytest` object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_type()`

Other typing expectation predicates: `expect_s3_class()`, `expect_s4_class()`

Examples

```
expect_type(1L, "integer") # Pass
expect_type(1.0, "integer") # Fail
```

expect_visible	<i>Expect Visible</i>
----------------	-----------------------

Description

Expect Visible

Usage

```
expect_visible(current, ..., info = NA_character_)
```

Arguments

current	[R object or expression] Object or expression to be tested
...	Ignored
info	[character(1L)] Optional user-defined message; must be a single string value

Value

A **tinytest** object. A tinytest object is a logical with attributes holding information about the test that was run

See Also

testthat equivalent: `testthat::expect_visible()`
`withVisible()`
 Other visibility expectation predicate: `expect_invisible()`

Examples

```
f <- function() invisible()
expect_visible(f()) # Fail

g <- function() NULL
expect_visible(g()) # Pass
```

skip

Stop Testing

Description

Unconditionally stop testing a **tinytest** test file, preventing additional tests in the file from running without triggering a failure. This is the low-level exit function for other `skip_()` functions in **tinyexpect**

Usage

```
skip(message = "Skipping")
```

Arguments

message	A message describing why the test file is being skipped
---------	---

Value

If called within a **tinytest** test, triggers an exit condition; otherwise, returns message

See Also

testthat equivalent: `testthat::skip()`

`tinytest::exit_file()`

Other "stop testing" functions: `skip_if_not_installed()`, `skip_on_bioc()`, `skip_on_ci()`, `skip_on_covr()`, `skip_on_cran()`, `skip_on_os()`

Examples

```
skip()
```

`skip_if_not_installed` *Stop Testing if a Required Package is Not Installed*

Description

Conditionally stop testing a **tinytest** test file if a required package is not available or not of a minimum required version

Usage

```
skip_if_not_installed(pkg, minimum_version = NULL, quietly = TRUE)
```

```
exit_if_not_installed(pkg, minimum_version = NULL, quietly = TRUE)
```

Arguments

<code>pkg</code>	Name of package to check for
<code>minimum_version</code>	Optional minimum required version of pkg
<code>quietly</code>	Attempt to find the package quietly; passed to <code>requireNamespace()</code>

Value

If called within a **tinytest** test and `pkg` is either not installed or not at least `minimum_version`, triggers an exit condition; otherwise, returns one of

- A string saying that `pkg` is not installed
- A string saying that `pkg` is installed, but not at least `minimum_version`
- `NULL` invisibly

“Skip” vs “Exit”

tinyexpect provides both “skip_” and “exit_” versions of “stop testing” functions due to the different philosophies of **tinytest** and **testthat**; in **testthat**, tests are encapsulated by **test_that()** to create smaller testing units within a single test file. As such, if a series of tests need to be passed over for some reason, it makes sense to “skip” a **test_that()** block and move on to the next block

tinytest, however, treats each test file as a testing unit. Each file in `inst/tinytest` is equivalent to a **testthat** **test_that()** block; as such, if a series of tests need to be passed over for some reason, it makes sense to “exit” a test file and move on to the next file in `inst/tinytest`

In order to provide compatibility with users transitioning from **testthat** to **tinytest**, and to provide continuity with the **tinytest** philosophy, **tinyexpect** provides both `skip_`- and `exit_`-prefixed “stop testing” functions that work identically to one another

See Also

testthat equivalent: `testthat::skip_if_not_installed()`

Other “stop testing” functions: `skip()`, `skip_on_bioc()`, `skip_on_ci()`, `skip_on_covr()`, `skip_on_cran()`, `skip_on_os()`

Examples

```
(pkg <- paste(sample(letters, size = 7L, replace = TRUE), collapse = ""))
skip_if_not_installed(pkg)

skip_if_not_installed("tinyexpect", minimum_version = "99.0.1")
```

skip_on_bioc

Stop Testing on the Bioconductor Build System

Description

Conditionally stop testing a **tinytest** test file if running on the Bioconductor Build System. This is determined by checking if the environment variable “IS_BIOC_BUILD_MACHINE” is set and is a non-`FALSE` value

Usage

```
skip_on_bioc()

exit_on_bioc()
```

Value

If called within a **tinytest** test running on the Bioconductor Build System, triggers an exit condition; otherwise, either a string saying “On Bioconductor” or `NULL` invisibly

See Also

testthat equivalent: `testthat::skip_on_bioc()`

Other "stop testing" functions: `skip()`, `skip_if_not_installed()`, `skip_on_ci()`, `skip_on_covr()`, `skip_on_cran()`, `skip_on_os()`

Examples

```
withr::with_envvar(c(IS_BIOC_BUILD_MACHINE = "true"), skip_on_bioc())
```

skip_on_ci

Stop Testing on CI

Description

Conditionally stop testing a **tinytest** test file if running on CI. This is determined by checking if the environment variable "CI" is set and is a non-FALSE value

Usage

```
skip_on_ci()
```

```
exit_on_ci()
```

Value

If called within a **tinytest** test running on CI, triggers an exit condition; otherwise, either a string saying "On CI" or NULL invisibly

"Skip" vs "Exit"

tinyexpect provides both "skip_" and "exit_" versions of "stop testing" functions due to the different philosophies of **tinytest** and **testthat**; in **testthat**, tests are encapsulated by `test_that()` to create smaller testing units within a single test file. As such, if a series of tests need to be passed over for some reason, it makes sense to "skip" a `test_that()` block and move on to the next block. **tinytest**, however, treats each test file as a testing unit. Each file in `inst/tinytest` is equivalent to a **testthat** `test_that()` block; as such, if a series of tests need to be passed over for some reason, it makes sense to "exit" a test file and move on to the next file in `inst/tinytest`.

In order to provide compatibility with users transitioning from **testthat** to **tinytest**, and to provide continuity with the **tinytest** philosophy, **tinyexpect** provides both `skip_`- and `exit_`-prefixed "stop testing" functions that work identically to one another.

See Also

testthat equivalent: `testthat::skip_on_ci()`

Other "stop testing" functions: `skip()`, `skip_if_not_installed()`, `skip_on_bioc()`, `skip_on_covr()`, `skip_on_cran()`, `skip_on_os()`

Examples

```
withr::with_envvar(c(CI = "true"), skip_on_ci())
```

skip_on_covr	Stop	Testing	Under	Rhref https://CRAN.R-
			covr	

Description

Conditionally stop testing a **tinytest** test file if running under **covr**. This is determined by checking if the environment variable “R_COVR” set and is a non-FALSE value

Usage

```
skip_on_covr()
```

```
exit_on_covr()
```

Value

If called within a **tinytest** test running under **covr**, triggers an exit condition; otherwise, either a string saying “On covr” or NULL invisibly

“Skip” vs “Exit”

tinyexpect provides both “skip_” and “exit_” versions of “stop testing” functions due to the different philosophies of **tinytest** and **testthat**; in **testthat**, tests are encapsulated by **test_that()** to create smaller testing units within a single test file. As such, if a series of tests need to be passed over for some reason, it makes sense to “skip” a **test_that()** block and move on to the next block. **tinytest**, however, treats each test file as a testing unit. Each file in `inst/tinytest` is equivalent to a **testthat** **test_that()** block; as such, if a series of tests need to be passed over for some reason, it makes sense to “exit” a test file and move on to the next file in `inst/tinytest`.

In order to provide compatibility with users transitioning from **testthat** to **tinytest**, and to provide continuity with the **tinytest** philosophy, **tinyexpect** provides both `skip_`- and `exit_`-prefixed “stop testing” functions that work identically to one another.

See Also

testthat equivalent: `testthat::skip_on_covr()`

Other “stop testing” functions: `skip()`, `skip_if_not_installed()`, `skip_on_bioc()`, `skip_on_ci()`, `skip_on_cran()`, `skip_on_os()`

Examples

```
withr::with_envvar(c(R_COVR = "true"), skip_on_covr())
```

skip_on_cran

*Stop Testing on CRAN***Description**

Conditionally stop testing a **tinytest** test file if running under **covr**. This is determined by checking if

- R is in a non-**interactive** session, and
- the environment variable “NOT_CRAN” is unset or is not a TRUE value

Usage

```
skip_on_cran()
```

```
exit_on_cran()
```

Value

If called within a **tinytest** test running on CRAN in a **non-interactive session**, triggers an exit condition; otherwise, either a string saying “On CRAN” or NULL invisibly

“Skip” vs “Exit”

tinyexpect provides both “skip_” and “exit_” versions of “stop testing” functions due to the different philosophies of **tinytest** and **testthat**; in **testthat**, tests are encapsulated by **test_that()** to create smaller testing units within a single test file. As such, if a series of tests need to be passed over for some reason, it makes sense to “skip” a **test_that()** block and move on to the next block. **tinytest**, however, treats each test file as a testing unit. Each file in `inst/tinytest` is equivalent to a **testthat** **test_that()** block; as such, if a series of tests need to be passed over for some reason, it makes sense to “exit” a test file and move on to the next file in `inst/tinytest`.

In order to provide compatibility with users transitioning from **testthat** to **tinytest**, and to provide continuity with the **tinytest** philosophy, **tinyexpect** provides both `skip_`- and `exit_`-prefixed “stop testing” functions that work identically to one another.

See Also

testthat equivalent: `testthat::skip_on_cran()`

Other “stop testing” functions: `skip()`, `skip_if_not_installed()`, `skip_on_bioc()`, `skip_on_ci()`, `skip_on_covr()`, `skip_on_os()`

Examples

```
withr::with_envvar(c(NOT_CRAN = "false"), skip_on_cran())
```

skip_on_os

Stop Testing on Specific Operating Systems and Architectures

Description

Conditionally stop testing a **tinystest** test file if running under a given operating system and/or architecture

Usage

```
skip_on_os(os, arch = NULL)
```

```
exit_on_os(os, arch = NULL)
```

Arguments

os	Operating system to not test on; choose one or more from: • “windows” • “mac” • “linux” • “solaris” The following OS designations are accepted as synonyms: • “darwin”: accepted in place of “mac” • “sunos”: accepted in place of “solaris” Pass TRUE to set os to all of the above; this is useful for limiting tests by architecture rather than OS/architecture combos
arch	Optional system architectures to not test on; note that this only applies to operating systems present in os

Value

If called within a **tinystest** test running under os and potentially on an arch system, triggers an exit condition; otherwise, returns one of

- A string saying that the code is running under os
- A string saying that the code is running under os on an arch system
- NULL invisibly

“Skip” vs “Exit”

tinyexpect provides both “skip_” and “exit_” versions of “stop testing” functions due to the different philosophies of **tinystest** and **testthat**; in **testthat**, tests are encapsulated by `test_that()` to create smaller testing units within a single test file. As such, if a series of tests need to be passed over for some reason, it makes sense to “skip” a `test_that()` block and move on to the next block

tinytest, however, treats each test file as a testing unit. Each file in `inst/tinytest` is equivalent to a **testthat** `test_that()` block; as such, if a series of tests need to be passed over for some reason, it makes sense to “exit” a test file and move on to the next file in `inst/tinytest`

In order to provide compatibility with users transitioning from **testthat** to **tinytest**, and to provide continuity with the **tinytest** philosophy, **tinyexpect** provides both `skip_`- and `exit_`-prefixed “stop testing” functions that work identically to one another

See Also

testthat equivalent: `testthat::skip_on_os()`

Tools for querying system OS and architecture: `Sys.info()`, `R.version[["arch"]]`

Other “stop testing” functions: `skip()`, `skip_if_not_installed()`, `skip_on_bioc()`, `skip_on_ci()`, `skip_on_covr()`, `skip_on_cran()`

Examples

```
(system <- tolower(Sys.info()[["sysname"]]))
skip_on_os(system)

# Nothing happens if on a different OS
(other <- sample(setdiff(c("windows", "mac", "linux", "solaris"), system), 1L))
skip_on_os(other)

# System architectures can be used to fine-tune skips
(sysarch <- R.version$arch)
skip_on_os(system, arch = sysarch)
```

Index

- * **condition**
 - expect_condition, 3
 - expect_no_condition, 9
- * **inheritance**
 - expect_s3_class, 10
 - expect_s4_class, 11
 - expect_type, 13
- * **set**
 - expect_contains, 4
 - expect_in, 5
 - expect_mapequal, 7
 - expect_setequal, 12
- * **skip**
 - skip, 14
 - skip_if_not_installed, 15
 - skip_on_bioc, 16
 - skip_on_ci, 17
 - skip_on_covr, 18
 - skip_on_cran, 19
 - skip_on_os, 20
- * **visibility**
 - expect_invisible, 6
 - expect_visible, 13

all.equal, 7

comparisons, 2

condition, 3, 9

exit_if_not_installed
(skip_if_not_installed), 15

exit_on_bioc (skip_on_bioc), 16

exit_on_ci (skip_on_ci), 17

exit_on_covr (skip_on_covr), 18

exit_on_cran (skip_on_cran), 19

exit_on_os (skip_on_os), 20

expect_comparisons, 2

expect_condition, 3, 10

expect_contains, 4, 5, 7, 12

expect_gt, 3

expect_gt (expect_comparisons), 2

expect_gte, 3

expect_gte (expect_comparisons), 2

expect_in, 5, 5, 7, 12

expect_invisible, 6, 14

expect_lt, 3

expect_lt (expect_comparisons), 2

expect_lte, 3

expect_lte (expect_comparisons), 2

expect_mapequal, 5, 7, 12

expect_named, 8

expect_no_condition, 4, 9

expect_s3_class, 10, 12, 13

expect_s4_class, 11, 11, 13

expect_setequal, 5, 7, 12

expect_type, 11, 12, 13

expect_visible, 6, 13

inherits, 10, 11

interactive, 19

invisible, 6

missing, 8

names, 8

non-interactive session, 19

R.version, 21

requireNamespace, 15

sets, 7, 12

skip, 14, 16–19, 21

skip_if_not_installed, 15, 15, 17–19, 21

skip_on_bioc, 15, 16, 16–19, 21

skip_on_ci, 15, 16, 17, 17–19, 21

skip_on_covr, 15–17, 18, 19, 21

skip_on_cran, 15–18, 19, 21

skip_on_os, 15–19, 20

Sys.info, 21

test_that, 16–21

testthat::expect_condition, [4](#)
testthat::expect_contains, [5](#)
testthat::expect_in, [5](#)
testthat::expect_invisible, [6](#)
testthat::expect_mapequal, [7](#)
testthat::expect_named, [8](#)
testthat::expect_no_condition, [10](#)
testthat::expect_s3_class, [11](#)
testthat::expect_s4_class, [12](#)
testthat::expect_setequal, [12](#)
testthat::expect_type, [13](#)
testthat::expect_visible, [14](#)
testthat::skip, [15](#)
testthat::skip_if_not_installed, [16](#)
testthat::skip_on_bioc, [17](#)
testthat::skip_on_ci, [17](#)
testthat::skip_on_covr, [18](#)
testthat::skip_on_cran, [19](#)
testthat::skip_on_os, [21](#)
tinytest, [2](#), [4–8](#), [10–14](#)
tinytest::exit_file, [15](#)
typeof, [13](#)

vector, [4](#), [5](#), [7](#), [12](#)

withVisible, [6](#), [14](#)