Package 'tinytest'

February 22, 2023

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

License GPL-3

Title Lightweight and Feature Complete Unit Testing Framework

Type Package

LazyLoad yes

Description Provides a lightweight (zero-dependency) and easy to use unit testing framework. Main features: install tests with the package. Test results are treated as data that can be stored and manipulated. Test files are R scripts interspersed with test commands, that can be programmed over. Fully automated build-install-test sequence for packages. Skip tests when not run locally (e.g. on CRAN). Flexible and configurable output printing. Compare computed output with output stored with the package. Run tests in parallel. Extensible by other packages. Report side effects.

Version 1.4.1

URL https://github.com/markvanderloo/tinytest

BugReports https://github.com/markvanderloo/tinytest/issues

Depends R (>= 3.0.0)

Imports parallel, utils

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author Mark van der Loo [aut, cre] (<https://orcid.org/0000-0002-9807-4686>)

Repository CRAN

Date/Publication 2023-02-22 00:40:02 UTC

R topics documented:

`at_home` *Detect not on CRANity*

Description

Detect whether we are running at home (i.e. not on CRAN, BioConductor, ...)

Usage

```
at_home()
```

Examples

```
# test will run locally, but not on CRAN
if ( at_home() ){
  expect_equal(2, 1+1)
}
```

`build_install_test` *build, install and test*

Description

Builds and installs the package in `pkgdir` under a temporary directory. Next, loads the package in a fresh R session and runs all the tests. For this function to work the following system requirements are necessary.

- `R CMD build` is available on your system
- `Rscript` is available on your system

Usage

```
build_install_test(
  pkgdir = "./",
  testdir = "tinytest",
  pattern = "^test.*\\.[rR]$",
  at_home = TRUE,
  verbose = getOption("tt.verbose", 2),
  color = getOption("tt.pr.color", TRUE),
  ncpu = 1,
  remove_side_effects = TRUE,
  side_effects = FALSE,
  lc_collate = getOption("tt.collate", NA),
  keep_tempdir = FALSE,
  encoding = "unknown"
)
```

Arguments

`pkgdir`	`[character]` Package directory
`testdir`	`[character]` Name of directory under `pkgdir/inst` containing test files.
`pattern`	`[character]` A regular expression that is used to find scripts in `dir` containing tests (by default `.R` or `.r` files starting with `test`).
`at_home`	`[logical]` toggle local tests.
`verbose`	`[logical]` toggle verbosity during execution
`color`	`[logical]` toggle colorize output
`ncpu`	`[numeric]` number of CPUs to use during the testing phase.
`remove_side_effects`	`[logical]` toggle remove user-defined side effects? See section on side effects.
`side_effects`	`[logical|list]` Either a logical, or a list of arguments to pass to `report_side_effects`.
`lc_collate`	`[character]` Locale setting used to sort the test files into the order of execution. The default `NA` ensures current locale is used. Set this e.g. to `"C"` to ensure bytewise and more platform-independent sorting (see details in `run_test_dir`.

keep_tempdir	[logical] keep directory where the pkg is installed and where tests are run? If TRUE, the directory is not deleted and it's location is printed.
encoding	[character] Encoding parameter passed to parse.

Value

A tinytests object.

See Also

Other test-files: exit_file(), run_test_dir(), run_test_file(), summary.tinytests(), test_package()

Examples

```
## Not run:
  ## If your package source directory is "./pkg" you can run
  build_install_test("pkg")

## End(Not run)
```

---

exit_file *Stop testing (conditionally)*

Description

Use exit_file to exit a file with a custom message, or use exit_if to exit if one or more conditions are met. exit_if will create a message akin to messages created by stopifnot.

Usage

```
exit_file(msg = "")

exit_if_not(...)
```

Arguments

msg	[character] An optional message to print after exiting.
...	A comma-separated list of conditions.

Value

The exit message

See Also

Other test-files: build_install_test(), run_test_dir(), run_test_file(), summary.tinytests(), test_package()

Examples

```
exit_file("I'm too tired to test")
exit_if_not(packageVersion("tinytest")  >= "1.0.0")
## Not run:
exit_if_not(requireNamespace("foo",quietly=TRUE))

## End(Not run)
```

`expect_equal` *Express expectations*

Description

Express expectations

Usage

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

expect_identical(current, target, info = NA_character_)

expect_equivalent(
  current,
  target,
  tolerance = sqrt(.Machine$double.eps),
  info = NA_character_,
  ...
)

expect_true(current, info = NA_character_)

expect_false(current, info = NA_character_)

expect_silent(current, quiet = TRUE, info = NA_character_)

expect_null(current, info = NA_character_)

expect_inherits(current, class, info = NA_character_)

expect_error(
  current,
```

```
  pattern = ".*",
  class = "error",
  info = NA_character_,
  ...
)

expect_warning(
  current,
  pattern = ".*",
  class = "warning",
  info = NA_character_,
  strict = FALSE,
  ...
)

expect_message(
  current,
  pattern = ".*",
  class = "message",
  info = NA_character_,
  strict = FALSE,
  ...
)

expect_stdout(current, pattern = ".*", info = NA_character_, ...)
```

Arguments

current	[R object or expression] Outcome or expression under scrutiny.
target	[R object or expression] Expected outcome
tolerance	[numeric] Test equality to machine rounding. Passed to all.equal (tolerance)
info	[character] scalar. Optional user-defined message. Must be a single character string. Multiline comments may be separated by "\n".
...	passed on to grepl (useful for e.g. fixed=TRUE).
quiet	[logical] suppress output printed by the current expression (see examples)
class	[character] For condition signals (error, warning, message) the class from which the condition should inherit.
pattern	[character] A regular expression to match the message.
strict	[logical] scalar. If set to TRUE, any exception worse than the wanted exception will cause the test to fail.

Details

expect_equivalent calls expect_equal with the extra arguments check.attributes=FALSE and use.names=FALSE

expect_silent fails when an error or warning is thrown.

expect_inherits fails when inherits(current,class) returns FALSE

expect_stdout Expects that output is written to stdout, for example using cat or print. Use pattern to specify a regular expression matching the output.

Value

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

More information and examples

- An overview of tinytest can be found in vignette("using_tinytest").
- Examples of how tinytest is used in practice can be found in vignette("tinytest_examples")

Note

Each expect_haha function can also be called as checkHaha. Although the interface is not entirely the same, it is expected that this makes migration from the RUnit framework a little easier, for those who wish to do so.

expect_error, expect_warning and expect_message will concatenate all messages when multiple exceptions are thrown, before matching the message to pattern.

When speccifying regular expression patterns for errors, warnings or messages, note that message adds a LF character by default at the end of the message string.

See Also

Other test-functions: expect_equal_to_reference(), expect_length(), expect_match(), ignore()

Examples

```
expect_equal(1 + 1, 2)       # TRUE
expect_equal(1 - 1, 2)       # FALSE
expect_equivalent(2, c(x=2)) # TRUE
expect_equal(2, c(x=2))      # FALSE


expect_silent(1+1)           # TRUE
expect_silent(1+"a")         # FALSE
expect_silent(print("hihi")) # TRUE, nothing goes to screen
expect_silent(print("hihi"), quiet=FALSE) # TRUE, and printed
```

expect_equal_to_reference *Compare object with object stored in a file*

Description

Compares the current value with a value stored to file with saveRDS. If the file does not exist, the current value is stored into file, and the test returns expect_null(NULL).

Usage

```
expect_equal_to_reference(current, file, ...)

expect_equivalent_to_reference(current, file, ...)
```

Arguments

current	[R object or expression] Outcome or expression under scrutiny.
file	[character] File where the target is stored. If file does not exist, current will be stored there.
...	passed to expect_equal, respectively expect_equivalent.

Note

Be aware that on CRAN it is not allowed to write data to user space. So make sure that the file is either stored with your tests, or generated with tempfile, or the test is skipped on CRAN, using at_home.

build_install_test clones the package and builds and tests it in a separate R session in the background. This means that if you create a file located at tempfile() during the run, this file is destroyed when the separate R session is closed.

See Also

Other test-functions: expect_equal(), expect_length(), expect_match(), ignore()

Examples

```
filename <- tempfile()
# this gives TRUE: the file does not exist, but is created now.
expect_equal_to_reference(1, file=filename)
# this gives TRUE: the file now exists, and its contents is equal
# to the current value
expect_equal_to_reference(1, file=filename)
# this gives FALSE: the file exists, but is contents is not equal
# to the current value
expect_equal_to_reference(2, file=filename)
```

expect_length *Check length of an object*

Description

Check length of an object

Usage

```
expect_length(current, length, info = NA_character_, ...)
```

Arguments

current	[object] An R object with a length
length	[integer] A nonnegative integer
info	[character] scalar. Optional user-defined message. Must be a single character string. Multiline comments may be separated by
...	Currently not used.

See Also

Other test-functions: expect_equal_to_reference(), expect_equal(), expect_match(), ignore()

Examples

```
expect_length(3:4, 2) # TRUE
expect_length(2:5, 1) # FALSE
```

expect_match *Match strings to a regular expression*

Description

Results in TRUE only when all elements of current match the regular expression in pattern. Matching is done by grepl.

Usage

```
expect_match(current, pattern, info = NA_character_, ...)
```

Arguments

current	[character] String(s) to check for pattern.
pattern	[character] A regular expression.
info	[character] scalar. Optional user-defined message. Must be a single character string. Multiline comments may be separated by "\n".
...	passed to grepl

See Also

Other test-functions: expect_equal_to_reference(), expect_equal(), expect_length(), ignore()

Examples

```
expect_match("hello world", "world")                    # TRUE
expect_match("hello world", "^world$")                  # FALSE
expect_match("HelLO woRlD", "world", ignore.case=TRUE)  # TRUE
expect_match(c("apple","banana"), "a")                  # TRUE
expect_match(c("apple","banana"), "b")                  # FALSE
```

---

format.tinytest *Print a tinytest object*

Description

Print a tinytest object

Usage

```
## S3 method for class 'tinytest'
format(x, type = c("long", "short"), ...)

## S3 method for class 'tinytest'
print(x, ...)
```

Arguments

x	A tinytest object
type	[logical] Toggle format type
...	passed to format.tinytest

Value

A character string

Examples

```
tt <- expect_equal(1+1, 3)
format(tt,"long")
format(tt,"short")
print(expect_equal(1+1, 2))
print(expect_equal(1+1, 3), type="long")
```

---

get_call_wd *Get workding dir from where a test was initiated*

Description

A test runner, like `run_test_file` changes R's working directory to the location of the test file temporarily while the tests run. This function can be used from within the test file to get R's working directory at the time `run_test_file` (or one of it's siblings) was called.

Usage

```
get_call_wd()
```

Value

`[character]` A path.

Examples

```
get_call_wd()
```

---

ignore *Ignore the output of an expectation*

Description

Ignored expectations are not reported in the test results. Ignoring is only useful for test files, and not for use directly at the command-line. See also the package vignette: `vignette("using_tinytest")`.

Usage

```
ignore(fun)
```

Arguments

`fun`	`[function]` An `expect_` function

Value

An ignored `function`

Details

`ignore` is a higher-order function: a function that returns another function. In particular, it accepts a function and returns a function that is almost identical to the input function. The only difference is that the return value of the function returned by `ignore` is not caught by `run_test_file` and friends. For example, `ignore(expect_true)` is a function, and we can use it as `ignore(expect_true)(1 == 1)`. The return value of `ignore(expect_true)(1==1)` is exactly the same as that for `expect_true(1==1)`.

See Also

Other test-functions: `expect_equal_to_reference()`, `expect_equal()`, `expect_length()`, `expect_match()`

Examples

```
## The result of 'expect_warning' is not stored in the test result when
## this is run from a file.
expect_true( ignore(expect_warning)(warning("foo!")) )
## Note the placement of the brackets in ignore(expect_warning)(...).
```

---

register_tinytest_extension

Register or unregister extension functions

Description

Functions to use in `.onLoad` and `.onUnload` by packages that extend **tinytest**.

Usage

```
register_tinytest_extension(pkg, functions)
```

Arguments

`pkg`	`[character]` scalar. Name of the package providing extensions.
`functions`	`[character]` vector. Name of the functions in the package that must be added.

The tinytest API

Packages can extend **tinytest** with expectation functions *if and only if* the following requirements are satisfied.

1. The extending functions return a `tinytest` object. This can be created by calling `tinytest()` with the arguments (defaults, if any, are in brackets):
   - `result`: A `logical` scalar: `TRUE` or `FALSE` (not `NA`)
   - `call`: The `call` to the expectation function. Usually the result of `sys.call(sys.parent(1))`
   - `diff` (`NA_character_`): A `character` scalar, with a long description of the difference. Sentences may be separated by `"\n"`.
   - `short` (`NA_character_`): Either `"data"`, if the difference is in the data. `"attr"` when attributes differ or `"xcpt"` when an expectation about an exception is not met. If there are differences in data and in attributes, the attributes take precedence.
   - `info` (`NA_character_`): A user-defined message.

   Observe that this requires the extending package to add **tinytest** to the `Imports` field in the package's `DESCRIPTION` file (this also holds for the following requirement).
2. Functions are registered in `.onLoad()` using `register_tinytest_extension()`. Functions that are already registered, including **tinytest** functions will be overwritten.

It is *recommended* to:

1. Follow the syntax conventions of **tinytest** so expectation functions start with `expect_`.
2. Explain to users of the extension package how to use the extension (see `using`).
3. include an `info` argument to `expect_` functions that is passed to `tinytest()`.

Minimal example packages

- Extending **tinytest**: tinytest.extension.
- Using a **tinytest** extension: using.tinytest.extension.

See Also

Other extensions: `tinytest()`, `using()`

`report_side_effects` *Report side effects for expressions in test files*

Description

Call this function from within a test file to report side effects.

Usage

```
report_side_effects(
  report = TRUE,
  envvar = report,
  pwd = report,
  files = report,
  locale = report
)
```

Arguments

report	[logical] report all side-effects
envvar	[logical] changes in environment variables
pwd	[logical] changes in working directory
files	[logical] changes in files in the directory where the test file lives. Also watches subdirectories.
locale	[logical] Changes in locale settings as detected by link[base]{Sys.getlocale} are reported.

Value

A named logical, indicating which aspects of the environment are tracked, invisibly.

Details

A side effect causes a change in an external variable outside of the scope of a function, or test file. This includes environment variables, global options, global R variables, creating files or directories, and so on.

If this function is called in a test file, side effects are monitored from that point in the file and only for that file. The state of the environment before and after running every expression in the file are compared.

There is some performance penalty in tracking external variables, especially for those that require a system call.

Note

There could be side-effects that are untrackable by **tinytest**. This includes packages that use a global internal state within their namespace or packages that use a global state within compiled code.

Examples

```
# switch on
report_side_effects()
# switch off
report_side_effects(FALSE)

# only report changes in environment variables
report_side_effects(report=FALSE, envvar=TRUE)
```

`run_test_dir` *Run all tests in a directory*

Description

`run\_test\_dir` runs all test files in a directory.

`test_all` is a convenience function for package development, that wraps `run_test_dir`. By default, it runs all files starting with `test` in `./inst/tinytest/`. It is assumed that all functions to be tested are loaded.

Usage

```
run_test_dir(
  dir = "inst/tinytest",
  pattern = "^test.*\\.[rR]$",
  at_home = TRUE,
  verbose = getOption("tt.verbose", 2),
  color = getOption("tt.pr.color", TRUE),
  remove_side_effects = TRUE,
  cluster = NULL,
  lc_collate = getOption("tt.collate", NA),
  ...
)

test_all(pkgdir = "./", testdir = "inst/tinytest", ...)
```

Arguments

`dir`	`[character]` path to directory
`pattern`	`[character]` A regular expression that is used to find scripts in `dir` containing tests (by default `.R` or `.r` files starting with `test`).
`at_home`	`[logical]` toggle local tests.
`verbose`	`[logical]` toggle verbosity during execution
`color`	`[logical]` toggle colorize output
`remove_side_effects`	`[logical]` toggle remove user-defined side effects. Environment variables (`Sys.setenv()`) and options (`options()`) defined in a test file are reset before running the next test file (see details).
`cluster`	A `makeCluster` object.
`lc_collate`	`[character]` Locale setting used to sort the test files into the order of execution. The default `NA` ensures current locale is used. Set this e.g. to `"C"` to ensure bytewise and more platform-independent sorting (see details).
`...`	Arguments passed to `run_test_file`
`pkgdir`	`[character]` scalar. Root directory of the package (i.e. direcory where `DESCRIPTION` and `NAMESPACE` reside).
`testdir`	`[character]` scalar. Subdirectory where test files are stored.

Value

A `tinytests` object

Details

We cannot guarantee that files will be run in any particular order accross all platforms, as it depends on the available collation charts (a chart that determines how alphabets are sorted). For this reason it is a good idea to create test files that run independent of each other so their order of execution does not matter. In tinytest, test files cannot share variables. The default behavior of test runners further discourages interdependence by resetting environment variables and options that are set in a test file after the file is executed. If an environment variable needs to survive a single file, use `base::Sys.setenv()` explicitly. Similarly, if an option setting needs to survive, use `base::options()`

Parallel tests

If `inherits(cluster, "cluster")` the tests are paralellized over a cluster of worker nodes. **tinytest** will be loaded onto each cluster node. All other preparation, including loading code from the tested package, must be done by the user. It is also up to the user to clean up the cluster after running tests. See the 'using tinytest' vignette for examples: `vignette("using_tinytest")`.

See Also

`makeCluster`, `clusterEvalQ`, `clusterExport`

Other test-files: `build_install_test()`, `exit_file()`, `run_test_file()`, `summary.tinytests()`, `test_package()`

Examples

```
# create a test file in tempdir
tests <- "
addOne <- function(x) x + 2

expect_true(addOne(0) > 0)
expect_equal(2, addOne(1))
"
testfile <- tempfile(pattern="test_", fileext=".R")
write(tests, testfile)

# extract testdir
testdir <- dirname(testfile)
# run all files starting with 'test' in testdir
out <- run_test_dir(testdir)
print(out)
dat <- as.data.frame(out)
```

`run_test_file` *Run an R file containing tests; gather results*

Description

Run an R file containing tests; gather results

Usage

```
run_test_file(
  file,
  at_home = TRUE,
  verbose = getOption("tt.verbose", 2),
  color = getOption("tt.pr.color", TRUE),
  remove_side_effects = TRUE,
  side_effects = FALSE,
  set_env = list(),
  encoding = "unknown",
  ...
)
```

Arguments

`file`	`[character]` File location of a .R file.
`at_home`	`[logical]` toggle local tests.
`verbose`	`[integer]` verbosity level. 0: be quiet, 1: print status per file, 2: print status and increase counter after each test expression.
`color`	`[logical]` toggle colorize counts in verbose mode (see Note)
`remove_side_effects`	`[logical]` toggle remove user-defined side effects? See section on side effects.
`side_effects`	`[logical|list]` Either a logical, or a list of arguments to pass to `report_side_effects`.
`set_env`	`[named list]`. Key=value pairs of environment variables that will be set before the test file is run and reset afterwards. These are not counted as side effects of the code under scrutiny.
`encoding`	`[character]` Define encoding argument passed to `parse` when parsing `file`.
`...`	Currently unused

Details

In **tinytest**, a test file is just an R script where some or all of the statements express an `expectation`. `run_test_file` runs the file while gathering results of the expectations in a `tinytests` object.

The graphics device is set to `pdf(file=tempfile())` for the run of the test file.

Value

A `list` of class `tinytests`, which is a list of `tinytest` objects.

Side-effects caused by test code

All calls to `Sys.setenv` and `options` defined in a test file are captured and undone once the test file has run, if `remove_side_effects` is set to `TRUE`.

Tracking side effects

Certain side effects can be tracked, even when they are not explicitly evoked in the test file. See `report_side_effects` for side effects tracked by **tinytest**. Calls to `report_side_effects` within the test file overrule settings provided with this function.

Note

Not all terminals support ansi escape characters, so colorized output can be switched off. This can also be done globally by setting `options(tt.pr.color=FALSE)`. Some terminals that do support ansi escape characters may contain bugs. An example is the RStudio terminal (RStudio 1.1) running on Ubuntu 16.04 (and possibly other OSs).

See Also

`ignore`

Other test-files: `build_install_test()`, `exit_file()`, `run_test_dir()`, `summary.tinytests()`, `test_package()`

Examples

```
# create a test file, in temp directory
tests <- "
addOne <- function(x) x + 2

Sys.setenv(lolz=2)

expect_true(addOne(0) > 0)
expect_equal(2, addOne(1))

Sys.unsetenv('lolz')
"
testfile <- tempfile(pattern="test_", fileext=".R")
write(tests, testfile)

# run test file
out <- run_test_file(testfile,color=FALSE)
out
# print everything in short format, include passes in print.
print(out, nlong=0, passes=TRUE)

# run test file, track supported side-effects
```

```
run_test_file(testfile, side_effects=TRUE)

# run test file, track only changes in working directory
run_test_file(testfile, side_effects=list(pwd=TRUE, envvar=FALSE))
```

`setup_tinytest` *Add tinytest to package source directory*

Description

Creates `inst/tinytest`, and an example test file in that directory. Creates `tests/tinytest.R` so the package is tested with `R CMD check`. Adds `tinytests` as a suggested package to the `DESCRIPTION`.

Usage

```
setup_tinytest(pkgdir, force = FALSE, verbose = TRUE)
```

Arguments

`pkgdir`	`[character]` Package source directory
`force`	`[logical]` Toggle overwrite existing files? (not folders)
`verbose`	`[logical]` Toggle print progress

Value

`NULL`, invisibly.

Note on `DESCRIPTION`

Fails when it does not exist. It is assumed that the package is named in the `DESCRIPTION`.

Examples

```
## Not run:
# an easy way to set up a package 'haha' that passes
# R CMD check
pkgKitten::kitten("haha")
tinytest::setup_tinytest("haha")

## End(Not run)
```

summary.tinytests *Tinytests object*

Description

An object of class `tinytests` (note: plural) results from running multiple tests from script. E.g. by running `run_test_file`.

Usage

```
## S3 method for class 'tinytests'
summary(object, ...)

all_pass(x)

any_pass(x)

all_fail(x)

any_fail(x)

## S3 method for class 'tinytests'
x[i]

## S3 method for class 'tinytests'
print(
  x,
  passes = getOption("tt.pr.passes", FALSE),
  sidefx = getOption("tt.pr.sidefx", TRUE),
  limit = getOption("tt.pr.limit", 7),
  nlong = getOption("tt.pr.nlong", 3),
  ...
)

## S3 method for class 'tinytests'
as.data.frame(x, ...)
```

Arguments

`object`	a `tinytests` object
`...`	passed to `format.tinytest`
`x`	a `tinytests` object
`i`	an index
`passes`	`[logical]` Toggle: print passing tests?
`sidefx`	`[logical]` Toggle: print side effects?

limit	[numeric] Max number of results to print
nlong	[numeric] First nlong results are printed in long format.

Value

For summary a table object

For all_pass, any_pass, all_fail, any_fail: a single logical

For `[.tinytests` a tinytests object.

For as.data.frame. a data frame.

Details

By default, the first 3 failing test results are printed in long form, the next 7 failing test results are printed in short form and all other failing tests are not printed. These defaults can be changed by passing options to print.tinytest, or by setting one or more of the following global options:

- tt.pr.passes Set to TRUE to print output of non-failing tests.
- tt.pr.limit Max number of results to print (e.g. Inf)
- tt.pr.nlong The number of results to print in long format (e.g. Inf).

For example, set options(tt.pr.limit=Inf) to print all test results. Furthermore, there is the option

- tt.pr.color,

which determines whether colored output is printed. If R is running in a dumb terminal (detected by comparing environment variable "TERM" to "dumb"), then this option is set to FALSE when the package is loaded.

See Also

Other test-files: build_install_test(), exit_file(), run_test_dir(), run_test_file(), test_package()

Examples

```
# create a test file in tempdir
tests <- "
addOne <- function(x) x + 2

expect_true(addOne(0) > 0)
expect_equal(2, addOne(1))
"
testfile <- tempfile(pattern="test_", fileext=".R")
write(tests, testfile)

# extract testdir
testdir <- dirname(testfile)
# run all files starting with 'test' in testdir
out <- run_test_dir(testdir)
```

```
#
# print results
print(out)
summary(out)
dat <- as.data.frame(out)
out[1]
```

`test_package` *Test a package during R CMD check or after installation*

Description

Run all tests in an installed package. Throw an error and print all failed test results when one or more tests fail if not in interactive mode (e.g. when R CMD check tests a package). This function is intended to be used by `R CMD check` or by a user that installed a package that uses the **tinytest** test infrastructure.

Usage

```
test_package(
  pkgname,
  testdir = "tinytest",
  lib.loc = NULL,
  at_home = FALSE,
  ncpu = NULL,
  ...
)
```

Arguments

`pkgname`	`[character]` scalar. Name of the package, as in the `DESCRIPTION` file.
`testdir`	`[character]` scalar. Path to installed directory. By default tinytest assumes that test files are in `inst/tinytest/`, which means that after installation and thus during `R CMD check` they are in `tinytest/`. See details for using alternate paths.
`lib.loc`	`[character]` scalar. location where the package is installed.
`at_home`	`[logical]` scalar. Are we at home? (see Details)
`ncpu`	A positive integer, or a `makeCluster` object.
`...`	extra arguments passed to `run_test_dir` (e.g. `ncpu`).

Value

If `interactive()`, a `tinytests` object. If not `interactive()`, an error is thrown when at least one test fails.

Details

We set `at_home=FALSE` by default so `R CMD check` will run the same as at CRAN. See the package vignette (Section 4) for tips on how to set up the package structure. `vignette("using_tinytest",package="tinytest")`.

Package authors who want to avoid installing tests with the package can create a directory under `tests`. If the test directoy is called `"tests/foo"`, use `test_package("pkgname", testdir="foo")` in `tests/tinytest.R`.

See Also

`setup_tinytest`

Other test-files: `build_install_test()`, `exit_file()`, `run_test_dir()`, `run_test_file()`, `summary.tinytests()`

Examples

```
## Not run:
# Create a file with the following content, to use
# tinytest as your unit testing framework:
  if (requireNamespace("tinytest", quietly=TRUE))
    tinytest::test_package("your package name")

## End(Not run)
```

---

using *Use an extension package.*

Description

Loads and attaches a package to the search path, and picks up the **tinytest** extension functions registered by the package. Package authors *must* call this function in *every* test file where an extension is used, or otherwise results from the extension package are not recorded (without a warning). Calling `using` in every file where an extension is used also ensures that tests can be run in parallel.

Usage

```
using(package, quietly = TRUE)
```

Arguments

`package`	the name of the extension package, given as name or character string.
`quietly`	Passed to `require`.

Value

A named `list`, with the package name and the names of the functions registered by `package` to extend **tinytest**. A message is emitted when the package registers no extension functions.

See Also

Other extensions: register_tinytest_extension(), tinytest()

Examples

```
## Not run:
  # In interactive session: see which functions are exported
  # by checkmate.tinytest
  out <- using(checkmate.tinytest)
  print(out)

## End(Not run)
```

Index