

# Package ‘withr’

May 8, 2026

**Title** Run Code 'With' Temporarily Modified Global State

**Version** 3.0.2

**Description** A set of functions to run code 'with' safely and temporarily modified global state. Many of these functions were originally a part of the 'devtools' package, this provides a simple package with limited dependencies to provide access to these functions.

**License** MIT + file LICENSE

**URL** <https://withr.r-lib.org>, <https://github.com/r-lib/withr#readme>

**BugReports** <https://github.com/r-lib/withr/issues>

**Depends** R (>= 3.6.0)

**Imports** graphics, grDevices

**Suggests** callr, DBI, knitr, methods, rlang, rmarkdown (>= 2.12),  
RSQLite, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Config/Needs/website** tidyverse/tidytemplate

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Collate** 'aaa.R' 'collate.R' 'connection.R' 'db.R' 'defer-exit.R'  
'standalone-defer.R' 'defer.R' 'devices.R' 'local\_.R' 'with\_.R'  
'dir.R' 'env.R' 'file.R' 'language.R' 'libpaths.R' 'locale.R'  
'makevars.R' 'namespace.R' 'options.R' 'par.R' 'path.R' 'rng.R'  
'seed.R' 'wrap.R' 'sink.R' 'tempfile.R' 'timezone.R'  
'torture.R' 'utils.R' 'with.R'

**NeedsCompilation** no

**Author** Jim Hester [aut],  
Lionel Henry [aut, cre],  
Kirill Müller [aut],  
Kevin Ushey [aut],  
Hadley Wickham [aut],

Winston Chang [aut],  
 Jennifer Bryan [ctb],  
 Richard Cotton [ctb],  
 Posit Software, PBC [cph, fnd]

**Maintainer** Lionel Henry <lionel@posit.co>

**Repository** CRAN

**Date/Publication** 2024-10-28 13:30:02 UTC

## Contents

defer . . . . .	2
devices . . . . .	4
withr . . . . .	6
with_collate . . . . .	8
with_connection . . . . .	9
with_db_connection . . . . .	10
with_dir . . . . .	11
with_envvar . . . . .	12
with_file . . . . .	13
with_gctorture2 . . . . .	14
with_language . . . . .	15
with_libpaths . . . . .	15
with_locale . . . . .	16
with_makevars . . . . .	18
with_options . . . . .	19
with_package . . . . .	20
with_par . . . . .	22
with_path . . . . .	24
with_rng_version . . . . .	25
with_seed . . . . .	26
with_sink . . . . .	27
with_tempfile . . . . .	28
with_temp_libpaths . . . . .	30
with_timezone . . . . .	31
<b>Index</b>	<b>33</b>

---

defer

*Defer Evaluation of an Expression*

---

## Description

Similar to `on.exit()`, but allows one to attach an expression to be evaluated when exiting any frame currently on the stack. This provides a nice mechanism for scoping side effects for the duration of a function's execution.

**Usage**

```
defer(expr, envir = parent.frame(), priority = c("first", "last"))

defer_parent(expr, priority = c("first", "last"))

deferred_run(envir = parent.frame())

deferred_clear(envir = parent.frame())
```

**Arguments**

expr	[expression] An expression to be evaluated.
envir	[environment] Attach exit handlers to this environment. Typically, this should be either the current environment or a parent frame (accessed through <code>parent.frame()</code> ).
priority	[character(1)] Specify whether this handler should be executed "first" or "last", relative to any other registered handlers on this environment.

**Details**

`defer()` works by attaching handlers to the requested environment (as an attribute called "handlers"), and registering an exit handler that executes the registered handler when the function associated with the requested environment finishes execution.

Deferred events can be set on the global environment, primarily to facilitate the interactive development of code that is intended to be executed inside a function or test. A message alerts the user to the fact that an explicit `deferred_run()` is the only way to trigger these deferred events. Use `deferred_clear()` to clear them without evaluation. The global environment scenario is the main motivation for these functions.

**Running handlers within `source()`**

withr handlers run within `source()` are run when `source()` exits rather than line by line.

This is only the case when the script is sourced in `globalenv()`. For a local environment, the caller needs to set `options(withr.hook_source = TRUE)`. This is to avoid paying the penalty of detecting `source()` in the normal usage of `defer()`.

**Examples**

```
# define a 'local' function that creates a file, and
# removes it when the parent function has finished executing
local_file <- function(path) {
  file.create(path)
  defer_parent(unlink(path))
}

# create tempfile path
path <- tempfile()
```

```

# use 'local_file' in a function
local({
  local_file(path)
  stopifnot(file.exists(path))
})

# file is deleted as we leave 'local' local
stopifnot(!file.exists(path))

# investigate how 'defer' modifies the
# executing function's environment
local({
  local_file(path)
  print(attributes(environment()))
})

# Note that examples lack function scoping so deferred calls are
# generally executed immediately
defer(print("one"))
defer(print("two"))

```

---

 devices

*Graphics devices*


---

### Description

Temporarily use a graphics device.

### Usage

```
with_bmp(new, code, ...)
```

```
local_bmp(new, ..., .local_envir = parent.frame())
```

```
with_cairo_pdf(new, code, ...)
```

```
local_cairo_pdf(new, ..., .local_envir = parent.frame())
```

```
with_cairo_ps(new, code, ...)
```

```
local_cairo_ps(new, ..., .local_envir = parent.frame())
```

```
with_pdf(new, code, ...)
```

```
local_pdf(new, ..., .local_envir = parent.frame())
```

```
with_postscript(new, code, ...)
```

```
local_postscript(new, ..., .local_envir = parent.frame())  
with_svg(new, code, ...)  
local_svg(new, ..., .local_envir = parent.frame())  
with_tiff(new, code, ...)  
local_tiff(new, ..., .local_envir = parent.frame())  
with_xfig(new, code, ...)  
local_xfig(new, ..., .local_envir = parent.frame())  
with_png(new, code, ...)  
local_png(new, ..., .local_envir = parent.frame())  
with_jpeg(new, code, ...)  
local_jpeg(new, ..., .local_envir = parent.frame())
```

### Arguments

<code>new</code>	[named character] New graphics device
<code>code</code>	[any] Code to execute in the temporary environment
<code>...</code>	Additional arguments passed to the graphics device.
<code>.local_envir</code>	[environment] The environment to use for scoping.

### Details

- `with_bmp()` and `local_bmp()` wrap around `grDevices::bmp()`.
- `with_cairo_pdf()` and `local_cairo_pdf()` wrap around `grDevices::cairo_pdf()`.
- `with_cairo_ps()` and `local_cairo_ps()` wrap around `grDevices::cairo_ps()`.
- `with_pdf()` and `local_pdf()` wrap around `grDevices::pdf()`.
- `with_postscript()` and `local_postscript()` wrap around `grDevices::postscript()`.
- `with_svg()` and `local_svg()` wrap around `grDevices::svg()`.
- `with_tiff()` and `local_tiff()` wrap around `grDevices::tiff()`.
- `with_xfig()` and `local_xfig()` wrap around `grDevices::xfig()`.
- `with_png()` and `local_png()` wrap around `grDevices::png()`.
- `with_jpeg()` and `local_jpeg()` wrap around `grDevices::jpeg()`.

**Value**

[any]

The results of the evaluation of the code argument.

**Functions**

- `with_bmp()`: BMP device
- `with_cairo_pdf()`: CAIRO\_PDF device
- `with_cairo_ps()`: CAIRO\_PS device
- `with_pdf()`: PDF device
- `with_postscript()`: POSTSCRIPT device
- `with_svg()`: SVG device
- `with_tiff()`: TIFF device
- `with_xfig()`: XFIG device
- `with_png()`: PNG device
- `with_jpeg()`: JPEG device

**See Also**

[withr](#) for examples

[Devices](#)

**Examples**

```
# dimensions are in inches
with_pdf(file.path(tempdir(), "test.pdf"), width = 7, height = 5,
  plot(runif(5))
)

# dimensions are in pixels
with_png(file.path(tempdir(), "test.png"), width = 800, height = 600,
  plot(runif(5))
)
```

---

withr

*Execute code in temporarily altered environment*

---

**Description**

All functions prefixed by `with_` work as follows. First, a particular aspect of the global environment is modified (see below for a list). Then, custom code (passed via the `code` argument) is executed. Upon completion or error, the global environment is restored to the previous state. Each `with_` function has a `local_` variant, which instead resets the state when the current evaluation context ends (such as the end of a function).

**Arguments pattern**

new	[various]	Values for setting
code	[any]	Code to execute in the temporary environment
...		Further arguments

**Usage pattern**

```
with_...(new, code, ...)
```

**withr functions**

- `with_collate()`: collation order
- `with_dir()`: working directory
- `with_envvar()`: environment variables
- `with_libpaths()`: library paths, replacing current libpaths
- `with_locale()`: any locale setting
- `with_makevars()`: Makevars variables
- `with_options()`: options
- `with_par()`: graphics parameters
- `with_path()`: PATH environment variable
- `with_sink()`: output redirection

**Creating new "with" functions**

All `with_` functions are created by a helper function, `with_()`. This function accepts two arguments: a setter function and an optional resetter function. The setter function is expected to change the global state and return an "undo instruction". This undo instruction is then passed to the resetter function, which changes back the global state. In many cases, the setter function can be used naturally as resetter.

**Author(s)**

**Maintainer:** Lionel Henry <lionel@posit.co>

Authors:

- Jim Hester
- Kirill Müller <kr1mlr+r@mailbox.org>
- Kevin Ushey <kevinushey@gmail.com>
- Hadley Wickham <hadley@posit.co>
- Winston Chang

Other contributors:

- Jennifer Bryan [contributor]
- Richard Cotton [contributor]
- Posit Software, PBC [copyright holder, funder]

## See Also

Useful links:

- <https://withr.r-lib.org>
- <https://github.com/r-lib/withr#readme>
- Report bugs at <https://github.com/r-lib/withr/issues>

## Examples

```
getwd()
with_dir(tempdir(), getwd())
getwd()

Sys.getenv("WITHR")
with_envvar(c("WITHR" = 2), Sys.getenv("WITHR"))
Sys.getenv("WITHR")

with_envvar(c("A" = 1),
  with_envvar(c("A" = 2), action = "suffix", Sys.getenv("A"))
)

# local variants are best used within other functions
f <- function(x) {
  local_envvar(c("WITHR" = 2))
  Sys.getenv("WITHR")
}
Sys.getenv("WITHR")
```

---

with\_collate

*Collation Order*

---

## Description

Temporarily change collation order by changing the value of the LC\_COLLATE locale.

## Usage

```
with_collate(new, code)

local_collate(new = list(), .local_envir = parent.frame())
```

**Arguments**

new	[character(1)] New collation order
code	[any] Code to execute in the temporary environment
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**Examples**

```
# Modify collation order:
x <- c("bernard", "b  r  nice", "b  atrice", "boris")

with_collate("fr_FR", sort(x))
#> [1] "b  atrice" "b  r  nice" "bernard" "boris"

with_collate("C", sort(x))
#> [1] "bernard" "boris" "b  atrice" "b  r  nice"
```

**See Also**

[withr](#) for examples

---

with_connection	<i>Connections which close themselves</i>
-----------------	---

---

**Description**

R file connections which are automatically closed.

**Usage**

```
with_connection(con, code)

local_connection(con, .local_envir = parent.frame())
```

**Arguments**

con	For with_connection() a named list with the connection(s) to create. For local_connection() the code to create a single connection, which is then returned.
-----	---

code	[any] Code to execute in the temporary environment
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
with_connection(list(con = file("foo", "w")), {
  writeLines(c("foo", "bar"), con)
})

read_foo <- function() {
  readLines(local_connection(file("foo", "r")))
}
read_foo()

unlink("foo")
```

---

with\_db\_connection      *DBMS Connections which disconnect themselves.*

---

**Description**

Connections to Database Management Systems which automatically disconnect. In particular connections which are created with `DBI::dbConnect()` and closed with `DBI::dbDisconnect()`.

**Usage**

```
with_db_connection(con, code)

local_db_connection(con, .local_envir = parent.frame())
```

**Arguments**

con	For <code>with_db_connection()</code> a named list with the connection(s) to create. For <code>local_db_connection()</code> the code to create a single connection, which is then returned.
code	[any] Code to execute in the temporary environment
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
db <- tempfile()
with_db_connection(
  list(con = DBI::dbConnect(RSQLite::SQLite(), db)), {
    DBI::dbWriteTable(con, "mtcars", mtcars)
  })

head_db_table <- function(...) {
  con <- local_db_connection(DBI::dbConnect(RSQLite::SQLite(), db))
  head(DBI::dbReadTable(con, "mtcars"), ...)
}
head_db_table()
unlink(db)
```

---

with\_dir

*Working directory*


---

**Description**

Temporarily change the current working directory.

**Usage**

```
with_dir(new, code)

local_dir(new = list(), .local_envir = parent.frame())
```

**Arguments**

new	[character(1)] New working directory
code	[any] Code to execute in the temporary environment
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples  
[setwd\(\)](#)

**Examples**

```
getwd()

with_dir(tempdir(), getwd())
```

---

with_envvar	<i>Environment variables</i>
-------------	------------------------------

---

**Description**

Temporarily change system environment variables.

**Usage**

```
with_envvar(new, code, action = "replace")

local_envvar(
  .new = list(),
  ...,
  action = "replace",
  .local_envir = parent.frame()
)
```

**Arguments**

new, .new	[named character] New environment variables
code	[any] Code to execute in the temporary environment
action	should new values "replace", "prefix" or "suffix" existing variables with the same name.
...	Named arguments with new environment variables.
.local_envir	[environment] The environment to use for scoping.

**Details**

if NA is used those environment variables will be unset. If there are any duplicated variable names only the last one is used.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples  
[Sys.setenv\(\)](#)

**Examples**

```
with_envvar(new = c("GITHUB_PAT" = "abcdef"), Sys.getenv("GITHUB_PAT"))

# with_envvar unsets variables after usage
Sys.getenv("TEMP_SECRET")
with_envvar(new = c("TEMP_SECRET" = "secret"), Sys.getenv("TEMP_SECRET"))
Sys.getenv("TEMP_SECRET")
```

---

with_file	<i>Files which delete themselves</i>
-----------	--------------------------------------

---

**Description**

Create files, which are then automatically removed afterwards.

**Usage**

```
with_file(file, code)

local_file(.file, ..., .local_envir = parent.frame())
```

**Arguments**

file, .file	[named list] Files to create.
code	[any] Code to execute in the temporary environment
...	Additional (possibly named) arguments of files to create.
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
with_file("file1", {
  writelines("foo", "file1")
  readlines("file1")
})

with_file(list("file1" = writeLines("foo", "file1")), {
  readlines("file1")
})
```

---

with_gctorture2	<i>Torture Garbage Collector</i>
-----------------	----------------------------------

---

**Description**

Temporarily turn gctorture2 on.

**Usage**

```
with_gctorture2(new, code, wait = new, inhibit_release = FALSE)
```

**Arguments**

new	[integer] run GC every 'step' allocations.
code	[any] Code to execute in the temporary environment
wait	integer; number of allocations to wait before starting GC torture.
inhibit_release	logical; do not release free objects for re-use: use with caution.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

---

with_language	<i>Language</i>
---------------	-----------------

---

**Description**

Temporarily change the language used for translations.

**Usage**

```
with_language(lang, code)
```

```
local_language(lang, .local_envir = parent.frame())
```

**Arguments**

lang	A BCP47 language code like "en" (English), "fr" (French), "fr_CA" (French Canadian). Formally, this is a lower case two letter <b>ISO 639 country code</b> , optionally followed by "_" or "-" and an upper case two letter <b>ISO 3166 region code</b> .
code	[any] Code to execute in the temporary environment
.local_envir	[environment] The environment to use for scoping.

**Examples**

```
with_language("en", try(mean[[1]]))
with_language("fr", try(mean[[1]]))
with_language("es", try(mean[[1]]))
```

---

with_libpaths	<i>Library paths</i>
---------------	----------------------

---

**Description**

Temporarily change library paths.

**Usage**

```
with_libpaths(new, code, action = "replace")
```

```
local_libpaths(new = list(), action = "replace", .local_envir = parent.frame())
```

**Arguments**

new	[character] New library paths
code	[any] Code to execute in the temporary environment
action	[character(1)] should new values "replace", "prefix" or "suffix" existing paths.
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

[.libPaths\(\)](#)

Other libpaths: [with\\_temp\\_libpaths\(\)](#)

**Examples**

```
.libPaths()  
new_lib <- tempfile()  
dir.create(new_lib)  
with_libpaths(new_lib, print(.libPaths()))  
unlink(new_lib, recursive = TRUE)
```

---

with\_locale

*Locale settings*

---

**Description**

Temporarily change locale settings.

**Usage**

```
with_locale(new, code)
```

```
local_locale(.new = list(), ..., .local_envir = parent.frame())
```

**Arguments**

new, .new	[named character] New locale settings
code	[any] Code to execute in the temporary environment
...	Additional arguments with locale settings.
.local_envir	[environment] The environment to use for scoping.

**Details**

Setting the LC\_ALL category is currently not implemented.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

[Sys.setlocale\(\)](#)

**Examples**

```
## Change locale for time:
df <- data.frame(
  stringsAsFactors = FALSE,
  date = as.Date(c("2019-01-01", "2019-02-01")),
  value = c(1, 2)
)
with_locale(new = c("LC_TIME" = "es_ES"), code = plot(df$date, df$value))
## Compare with:
# plot(df$date, df$value)

## Month names:
with_locale(new = c("LC_TIME" = "en_GB"), format(ISOdate(2000, 1:12, 1), "%B"))
with_locale(new = c("LC_TIME" = "es_ES"), format(ISOdate(2000, 1:12, 1), "%B"))

## Change locale for currencies:
with_locale(new = c("LC_MONETARY" = "it_IT"), Sys.localeconv())
with_locale(new = c("LC_MONETARY" = "en_US"), Sys.localeconv())

## Ordering:
x <- c("bernard", "b er nice", "b atrice", "boris")
with_locale(c(LC_COLLATE = "fr_FR"), sort(x))
with_locale(c(LC_COLLATE = "C"), sort(x))
```

with\_makevars

*Makevars variables***Description**

Temporarily change contents of an existing Makevars file.

**Usage**

```
with_makevars(
  new,
  code,
  path = makevars_user(),
  assignment = c("=", ":", "?=", "+=")
)

local_makevars(
  .new = list(),
  ...,
  .path = makevars_user(),
  .assignment = c("=", ":", "?=", "+="),
  .local_envir = parent.frame()
)
```

**Arguments**

new, .new	[named character] New variables and their values
code	[any] Code to execute in the temporary environment
path, .path	[character(1)] location of existing Makevars file to modify.
assignment, .assignment	[character(1)] assignment type to use.
...	Additional new variables and their values.
.local_envir	[environment] The environment to use for scoping.

**Details**

If no Makevars file exists or the fields in new do not exist in the existing Makevars file then the fields are added to the new file. Existing fields which are not included in new are appended unchanged. Fields which exist in Makevars and in new are modified to use the value in new.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
writeLines("void foo(int* bar) { *bar = 1; }\n", "foo.c")
system("R CMD SHLIB --preclean -c foo.c")
with_makevars(c(CFLAGS = "-O3"), system("R CMD SHLIB --preclean -c foo.c"))
unlink(c("foo.c", "foo.so"))
```

---

with_options	<i>Options</i>
--------------	----------------

---

**Description**

Temporarily change global options.

**Usage**

```
with_options(new, code)

local_options(.new = list(), ..., .local_envir = parent.frame())
```

**Arguments**

new, .new	[named list] New options and their values
code	[any] Code to execute in the temporary environment
...	Additional options and their values
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples  
[options\(\)](#)

**Examples**

```

# number of significant digits to print
getOption("digits")
# modify temporarily the number of significant digits to print
with_options(list(digits = 3), getOption("digits"))
with_options(list(digits = 3), print(pi))

# modify temporarily the character to be used as the decimal point
getOption("digits")
with_options(list(OutDec = ","), print(pi))

# modify temporarily multiple options
with_options(list(OutDec = ",", digits = 3), print(pi))

# modify, within the scope of the function, the number of
# significant digits to print
print_3_digits <- function(x) {
  # assign 3 to the option "digits" for the rest of this function
  # after the function exits, the option will return to its previous
  # value
  local_options(list(digits = 3))
  print(x)
}

print_3_digits(pi) # returns 3.14
print(pi)         # returns 3.141593

```

---

with\_package

*Execute code with a modified search path*


---

**Description**

with\_package() attaches a package to the search path, executes the code, then removes the package from the search path. The package namespace is *not* unloaded however. with\_namespace() does the same thing, but attaches the package namespace to the search path, so all objects (even unexported ones) are also available on the search path.

**Usage**

```

with_package(
  package,
  code,
  pos = 2,
  lib.loc = NULL,
  character.only = TRUE,
  logical.return = FALSE,
  warn.conflicts = FALSE,
  quietly = TRUE,

```

```

    verbose = getOption("verbose")
  )

  local_package(
    package,
    pos = 2,
    lib.loc = NULL,
    character.only = TRUE,
    logical.return = FALSE,
    warn.conflicts = FALSE,
    quietly = TRUE,
    verbose = getOption("verbose"),
    .local_envir = parent.frame()
  )

  with_namespace(package, code, warn.conflicts = FALSE)

  local_namespace(package, .local_envir = parent.frame(), warn.conflicts = FALSE)

  with_environment(
    env,
    code,
    pos = 2L,
    name = format(env),
    warn.conflicts = FALSE
  )

  local_environment(
    env,
    pos = 2L,
    name = format(env),
    warn.conflicts = FALSE,
    .local_envir = parent.frame()
  )

```

### Arguments

package	[character(1)] package name to load.
code	[any] Code to execute in the temporary environment
pos	the position on the search list at which to attach the loaded namespace. Can also be the name of a position on the current search list as given by <a href="#">search()</a> .
lib.loc	a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known to <a href="#">.libPaths()</a> . Non-existent library trees are silently ignored.
character.only	a logical indicating whether package or help can be assumed to be character strings.

logical.return	logical. If it is TRUE, FALSE or TRUE is returned to indicate success.
warn.conflicts	logical. If TRUE, warnings are printed about <a href="#">conflicts</a> from attaching the new package. A conflict is a function masking a function, or a non-function masking a non-function. The default is TRUE unless specified as FALSE in the <code>conflicts.policy</code> option.
quietly	a logical. If TRUE, no message confirming package attaching is printed, and most often, no errors/warnings are printed if package attaching fails.
verbose	a logical. If TRUE, additional diagnostics are printed.
.local_envir	[environment] The environment to use for scoping.
env	[environment()] Environment to attach.
name	name to use for the attached database. Names starting with <code>package:</code> are reserved for <a href="#">library</a> .

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
## Not run:
with_package("ggplot2", {
  ggplot(mtcars) + geom_point(aes(wt, hp))
})

## End(Not run)
```

---

with\_par

*Graphics parameters*


---

**Description**

Temporarily change graphics parameters.

**Usage**

```
with_par(new, code, no.readonly = FALSE)

local_par(
  .new = list(),
  ...,
  no.readonly = FALSE,
  .local_envir = parent.frame()
)
```

**Arguments**

new, .new	[named list] New graphics parameters and their values
code	[any] Code to execute in the temporary environment
no.readonly	[logical(1)] see <a href="#">par()</a> documentation.
...	Additional graphics parameters and their values.
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples  
[par\(\)](#)

**Examples**

```
old <- par("col" = "black")

# This will be in red
with_par(list(col = "red", pch = 19),
  plot(mtcars$hp, mtcars$wt)
)

# This will still be in black
plot(mtcars$hp, mtcars$wt)

par(old)
```

---

with_path	<i>PATH environment variable</i>
-----------	----------------------------------

---

### Description

Temporarily change the system search path.

### Usage

```
with_path(new, code, action = c("prefix", "suffix", "replace"))

local_path(
  new = list(),
  action = c("prefix", "suffix", "replace"),
  .local_envir = parent.frame()
)
```

### Arguments

new	[character] New PATH entries
code	[any] Code to execute in the temporary environment
action	[character(1)] Should new values "replace", "prefix" (the default) or "suffix" existing paths
.local_envir	[environment] The environment to use for scoping.

### Value

[any]  
The results of the evaluation of the code argument.

### See Also

[withr](#) for examples  
[Sys.setenv\(\)](#)

### Examples

```
# temporarily modify the system PATH, *prefixing* the current path
with_path(getwd(), Sys.getenv("PATH"))
# temporarily modify the system PATH, *appending* to the current path
with_path(getwd(), Sys.getenv("PATH"), "suffix")
```

---

with_rng_version	<i>RNG version</i>
------------------	--------------------

---

### Description

Change the RNG version and restore it afterwards.

### Usage

```
with_rng_version(version, code)
```

```
local_rng_version(version, .local_envir = parent.frame())
```

### Arguments

version [character(1)] an R version number, e.g. "3.5.0", to switch to the RNG this version of R uses. See [RNGversion\(\)](#).

code [any]  
Code to execute in the temporary environment

.local\_envir The environment to apply the change to.

### Details

with\_rng\_version() runs the code with the specified RNG version and resets it afterwards.

local\_rng\_version() changes the RNG version for the caller execution environment.

### Value

[any]  
The results of the evaluation of the code argument.

### See Also

[withr](#) for examples

[RNGversion\(\)](#), [RNGkind\(\)](#), [with\\_seed\(\)](#).

### Examples

```
RNGkind()  
with_rng_version("3.0.0", RNGkind())  
with_rng_version("1.6.0", RNGkind())
```

```
with_rng_version("3.0.0",  
  with_seed(42, sample(1:100, 3)))
```

```
with_rng_version("1.6.0",  
  with_seed(42, sample(1:100, 3)))
```

```

RNGkind()

fun1 <- function() {
  local_rng_version("3.0.0")
  with_seed(42, sample(1:100, 3))
}

fun2 <- function() {
  local_rng_version("1.6.0")
  with_seed(42, sample(1:100, 3))
}

RNGkind()
fun1()
fun2()
RNGkind()

```

---

with\_seed

*Random seed*


---

## Description

with\_seed() runs code with a specific random seed and resets it afterwards.

with\_preserve\_seed() runs code with the current random seed and resets it afterwards.

## Usage

```

with_seed(
  seed,
  code,
  .rng_kind = NULL,
  .rng_normal_kind = NULL,
  .rng_sample_kind = NULL
)

```

```

local_seed(
  seed,
  .local_envir = parent.frame(),
  .rng_kind = NULL,
  .rng_normal_kind = NULL,
  .rng_sample_kind = NULL
)

```

```
with_preserve_seed(code)
```

```
local_preserve_seed(.local_envir = parent.frame())
```

**Arguments**

seed [integer(1)]  
The random seed to use to evaluate the code.

code [any]  
Code to execute in the temporary environment

.rng\_kind, .rng\_normal\_kind, .rng\_sample\_kind  
[character(1)]  
Kind of RNG to use. Passed as the kind, normal.kind, and sample.kind arguments of `RNGkind()`.

.local\_envir [environment]  
The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

**Examples**

```
# Same random values:
with_preserve_seed(runif(5))
with_preserve_seed(runif(5))

# Use a pseudorandom value as seed to advance the RNG and pick a different
# value for the next call:
with_seed(seed <- sample.int(.Machine$integer.max, 1L), runif(5))
with_seed(seed, runif(5))
with_seed(seed <- sample.int(.Machine$integer.max, 1L), runif(5))
```

---

with\_sink

*Output redirection*

---

**Description**

Temporarily divert output to a file via `sink()`. For sinks of type message, an error is raised if such a sink is already active.

**Usage**

```
with_output_sink(new, code, append = FALSE, split = FALSE)

local_output_sink(
  new = list(),
```

```

    append = FALSE,
    split = FALSE,
    .local_envir = parent.frame()
)

with_message_sink(new, code, append = FALSE)

local_message_sink(new = list(), append = FALSE, .local_envir = parent.frame())

```

### Arguments

new	[character(1) connection] A writable <a href="#">connection</a> or a character string naming the file to write to. Passing NULL will throw an error.
code	[any] Code to execute in the temporary environment
append	logical. If TRUE, output will be appended to file; otherwise, it will overwrite the contents of file.
split	logical: if TRUE, output will be sent to the new sink and to the current output stream, like the Unix program tee.
.local_envir	[environment] The environment to use for scoping.

### Value

[any]  
The results of the evaluation of the code argument.

### See Also

[withr](#) for examples  
[sink\(\)](#)

---

with_tempfile	<i>Temporary files and directories</i>
---------------	--

---

### Description

Temporarily create a file or directory, which will automatically be deleted once you're finished with it.

### Usage

```

with_tempfile(
  new,
  code,
  envir = parent.frame(),

```

```

    .local_envir = parent.frame(),
    pattern = "file",
    tmpdir = tmpdir(),
    fileext = ""
  )

local_tempfile(
  new = NULL,
  lines = NULL,
  envir = parent.frame(),
  .local_envir = parent.frame(),
  pattern = "file",
  tmpdir = tmpdir(),
  fileext = ""
)

with_tmpdir(
  code,
  clean = TRUE,
  pattern = "file",
  tmpdir = tmpdir(),
  fileext = ""
)

local_tmpdir(
  pattern = "file",
  tmpdir = tmpdir(),
  fileext = "",
  .local_envir = parent.frame(),
  clean = TRUE
)

```

### Arguments

<code>new</code>	[character vector] (Deprecated for <code>local_tempfile()</code> ) Names of temporary file handles to create.
<code>code</code>	[any] Code to execute in the temporary environment
<code>envir</code>	[environment] Deprecated in favor of <code>.local_envir</code> .
<code>.local_envir</code>	[environment] The environment to use for scoping.
<code>pattern</code>	a non-empty character vector giving the initial part of the name.
<code>tmpdir</code>	a non-empty character vector giving the directory name.
<code>fileext</code>	a non-empty character vector giving the file extension.
<code>lines</code>	Optionally, supply a character vector of lines to be written to path. This is useful if you want to seed the file with some default content.

`clean` [logical(1)]  
A logical indicating if the temporary directory should be deleted after use (TRUE, default) or left alone (FALSE).

### Value

[any]  
The results of the evaluation of the code argument.

### See Also

[withr](#) for examples

### Examples

```
# local_tempfile() is the easiest to use because it returns a path
local({
  path1 <- local_tempfile(lines = c("x,y", "1,2"))
  readLines(path1)
})
# the file is deleted automatically
file.exists(path1)

# with_tempfile() is a bit trickier; the first argument gives the name
# of a variable that will contain the path:
with_tempfile("path2", {
  print(path2)
  write.csv(iris, path2)
  file.size(path2)
})

# Note that this variable is only available in the scope of with_tempfile
try(path2)
```

---

with\_temp\_libpaths     *Library paths*

---

### Description

Temporarily prepend a new temporary directory to the library paths.

### Usage

```
with_temp_libpaths(code, action = "prefix")

local_temp_libpaths(action = "prefix", .local_envir = parent.frame())
```

**Arguments**

code	[any] Code to execute in the temporary environment
action	[character(1)] should new values "replace", "prefix" or "suffix" existing paths.
.local_envir	[environment] The environment to use for scoping.

**Value**

[any]  
The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

[.libPaths\(\)](#)

Other libpaths: [with\\_libpaths\(\)](#)

---

with_timezone	<i>Time zone</i>
---------------	------------------

---

**Description**

Change the time zone, and restore it afterwards.

**Usage**

```
with_timezone(tz, code)
```

```
local_timezone(tz, .local_envir = parent.frame())
```

**Arguments**

tz	[character(1)] a valid time zone specification, note that time zone names might be platform dependent.
code	[any] Code to execute in the temporary environment
.local_envir	The environment to apply the change to.

**Details**

`with_timezone()` runs the code with the specified time zone and resets it afterwards.

`local_timezone()` changes the time zone for the caller execution environment.

**Value**

[any]

The results of the evaluation of the code argument.

**See Also**

[withr](#) for examples

[Sys.timezone\(\)](#).

**Examples**

```
Sys.time()
with_timezone("Europe/Paris", print(Sys.time()))
with_timezone("America/Los_Angeles", print(Sys.time()))
```

```
fun1 <- function() {
  local_timezone("CET")
  print(Sys.time())
}
```

```
fun2 <- function() {
  local_timezone("America/Los_Angeles")
  print(Sys.time())
}
Sys.time()
fun1()
fun2()
Sys.time()
```

# Index

- \* **libpaths**
  - with\_libpaths, 15
  - with\_temp\_libpaths, 30
- \* **local-related functions**
  - defer, 2
  - .libPaths, 21
  - .libPaths(), 16, 31
- conflicts, 22
- connection, 28
- defer, 2
- defer\_parent (defer), 2
- deferred\_clear (defer), 2
- deferred\_run (defer), 2
- Devices, 6
- devices, 4
- grDevices::bmp(), 5
- grDevices::cairo\_pdf(), 5
- grDevices::cairo\_ps(), 5
- grDevices::jpeg(), 5
- grDevices::pdf(), 5
- grDevices::png(), 5
- grDevices::postscript(), 5
- grDevices::svg(), 5
- grDevices::tiff(), 5
- grDevices::xfig(), 5
- library, 22
- local\_bmp (devices), 4
- local\_cairo\_pdf (devices), 4
- local\_cairo\_ps (devices), 4
- local\_collate (with\_collate), 8
- local\_connection (with\_connection), 9
- local\_db\_connection
  - (with\_db\_connection), 10
- local\_dir (with\_dir), 11
- local\_environment (with\_package), 20
- local\_envvar (with\_envvar), 12
- local\_file (with\_file), 13
- local\_jpeg (devices), 4
- local\_language (with\_language), 15
- local\_libpaths (with\_libpaths), 15
- local\_locale (with\_locale), 16
- local\_makevars (with\_makevars), 18
- local\_message\_sink (with\_sink), 27
- local\_namespace (with\_package), 20
- local\_options (with\_options), 19
- local\_output\_sink (with\_sink), 27
- local\_package (with\_package), 20
- local\_par (with\_par), 22
- local\_path (with\_path), 24
- local\_pdf (devices), 4
- local\_png (devices), 4
- local\_postscript (devices), 4
- local\_preserve\_seed (with\_seed), 26
- local\_rng\_version (with\_rng\_version), 25
- local\_seed (with\_seed), 26
- local\_svg (devices), 4
- local\_temp\_libpaths
  - (with\_temp\_libpaths), 30
- local\_tempdir (with\_tempfile), 28
- local\_tempfile (with\_tempfile), 28
- local\_tiff (devices), 4
- local\_timezone (with\_timezone), 31
- local\_xfig (devices), 4
- on.exit(), 2
- options(), 19
- par(), 23
- parent.frame(), 3
- RNGkind(), 25, 27
- RNGversion(), 25
- search, 21
- setwd(), 12
- sink(), 27, 28

`Sys.setenv()`, [13](#), [24](#)  
`Sys.setlocale()`, [17](#)  
`Sys.timezone()`, [32](#)

`with_()`, [7](#)  
`with_bmp (devices)`, [4](#)  
`with_cairo_pdf (devices)`, [4](#)  
`with_cairo_ps (devices)`, [4](#)  
`with_collate`, [8](#)  
`with_collate()`, [7](#)  
`with_connection`, [9](#)  
`with_db_connection`, [10](#)  
`with_dev (devices)`, [4](#)  
`with_device (devices)`, [4](#)  
`with_dir`, [11](#)  
`with_dir()`, [7](#)  
`with_environment (with_package)`, [20](#)  
`with_envvar`, [12](#)  
`with_envvar()`, [7](#)  
`with_file`, [13](#)  
`with_gctorture2`, [14](#)  
`with_jpeg (devices)`, [4](#)  
`with_language`, [15](#)  
`with_libpaths`, [15](#), [31](#)  
`with_libpaths()`, [7](#)  
`with_locale`, [16](#)  
`with_locale()`, [7](#)  
`with_makevars`, [18](#)  
`with_makevars()`, [7](#)  
`with_message_sink (with_sink)`, [27](#)  
`with_namespace (with_package)`, [20](#)  
`with_options`, [19](#)  
`with_options()`, [7](#)  
`with_output_sink (with_sink)`, [27](#)  
`with_package`, [20](#)  
`with_par`, [22](#)  
`with_par()`, [7](#)  
`with_path`, [24](#)  
`with_path()`, [7](#)  
`with_pdf (devices)`, [4](#)  
`with_png (devices)`, [4](#)  
`with_postscript (devices)`, [4](#)  
`with_preserve_seed (with_seed)`, [26](#)  
`with_rng_version`, [25](#)  
`with_seed`, [26](#)  
`with_seed()`, [25](#)  
`with_sink`, [27](#)  
`with_sink()`, [7](#)  
`with_svg (devices)`, [4](#)  
`with_temp_libpaths`, [16](#), [30](#)  
`with_tempdir (with_tempfile)`, [28](#)  
`with_tempfile`, [28](#)  
`with_tiff (devices)`, [4](#)  
`with_timezone`, [31](#)  
`with_xfig (devices)`, [4](#)  
`withr`, [6](#), [6](#), [9–14](#), [16](#), [17](#), [19](#), [22–25](#), [27](#), [28](#),  
[30–32](#)  
`withr-package (withr)`, [6](#)