Package 'zip'

June 10, 2026

Title Cross-Platform 'zip' Compression

Version 3.0.0

Description Cross-Platform 'zip' Compression Library. A replacement for the 'zip' function, that does not require any additional external tools on any platform.

License MIT + file LICENSE

URL https://github.com/r-lib/zip, https://r-lib.github.io/zip/

BugReports https://github.com/r-lib/zip/issues

LinkingTo cli

Suggests callr, cli, curl, pillar, processx, R6, testthat, webfakes, withr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first large-files, http

Config/usethis/last-upkeep 2025-05-07

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Gábor Csárdi [aut, cre],
Kuba Podgórski [ctb],
Rich Geldreich [ctb],
Arm Limited [ctb, cph] (bundled Mbed TLS crypto subset in src/mbedtls/ (Apache-2.0)),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2026-06-10 12:40:02 UTC

Contents

`deflate` *Compress a raw GZIP stream*

Description

Compress a raw GZIP stream

Usage

```
deflate(buffer, level = 6L, pos = 1L, size = NULL)
```

Arguments

`buffer`	Raw vector, containing the data to compress.
`level`	Compression level, integer between 1 (fatest) and 9 (best).
`pos`	Start position of data to compress in `buffer`.
`size`	Compressed size estimate, or `NULL`. If not given, or too small, the output buffer is resized multiple times.

Value

Named list with three entries:

- `output`: raw vector, the compressed data,
- `bytes_read`: number of bytes used from `buffer`,
- `bytes_written`: number of bytes written to the output buffer.

See Also

`base::memCompress()` does the same with `type = "gzip"`, but it does not tell you the number of bytes read from the input.

Examples

```
data_gz <- deflate(charToRaw("Hello world!"))
inflate(data_gz$output)
```

`inflate` *Uncompress a raw GZIP stream*

Description

Uncompress a raw GZIP stream

Usage

```
inflate(buffer, pos = 1L, size = NULL, raw = FALSE)
```

Arguments

`buffer`	Raw vector, containing the data to uncompress.
`pos`	Start position of data to uncompress in `buffer`.
`size`	Uncompressed size estimate, or `NULL`. If not given, or too small, the output buffer is resized multiple times.
`raw`	Whether `buffer` contains a raw DEFLATE stream, i.e. one without a zlib header and trailer. The default (`FALSE`) expects a zlib stream.

Value

Named list with three entries:

- `output`: raw vector, the uncompressed data,
- `bytes_read`: number of bytes used from `buffer`,
- `bytes_written`: number of bytes written to the output buffer.

See Also

`base::memDecompress()` does the same with `type = "gzip"`, but it does not tell you the number of bytes read from the input.

Examples

```
data_gz <- deflate(charToRaw("Hello world!"))
inflate(data_gz$output)
```

unzip *Uncompress 'zip' Archives*

Description

`unzip()` always restores modification times of the extracted files and directories.

Usage

```
unzip(
  zipfile,
  files = NULL,
  overwrite = TRUE,
  junkpaths = FALSE,
  exdir = ".",
  encoding = NULL,
  password = NULL
)
```

Arguments

`zipfile`	Path to the zip file to uncompress, or a character vector of paths. When multiple paths are given and all other arguments are at their defaults, the files are unzipped concurrently in a thread pool. Set the `zip_threads` option or the `ZIP_THREADS` environment variable to control the number of threads used. By default zip uses two threads.
`files`	Character vector of files to extract from the archive. Files within directories can be specified, but they must use a forward slash as path separator, as this is what zip files use internally. If `NULL`, all files will be extracted.
`overwrite`	Whether to overwrite existing files. If `FALSE` and a file already exists, then an error is thrown.
`junkpaths`	Whether to ignore all directory paths when creating files. If `TRUE`, all files will be created in `exdir`.
`exdir`	Directory to uncompress the archive to. If it does not exist, it will be created.
`encoding`	Encoding to use for entry filenames. ZIP files signal UTF-8 filenames via a flag in each entry; those are always decoded as UTF-8 regardless of `encoding`. For entries without that flag, `encoding` is used; `NULL` (the default) falls back to IBM CP437, which is what the ZIP specification prescribes for legacy entries. The value is passed to `iconv()`.
`password`	Password for decrypting encrypted entries. It can be a string, a raw vector, or a function that returns one of these. If `NULL` (the default), the `zip_password` option is used, or no password if that is also `NULL`. The password is silently ignored for entries that are not encrypted.

Value

A data frame with one row per extracted entry and columns, invisibly: `filename` (path within the archive), `compressed_size`, `uncompressed_size`, `timestamp`, `permissions`, `crc32`, `offset`, `type` (same as in `zip_list()`), and `path` (absolute path to the extracted file on disk).

Permissions

If the zip archive stores permissions and was created on Unix, the permissions will be restored.

See Also

Other zip/unzip functions: `zip_list()`

Examples

```
## temporary directory, to avoid messing up the user's workspace.
dir.create(tmp <- tempfile())
dir.create(file.path(tmp, "mydir"))
cat("first file", file = file.path(tmp, "mydir", "file1"))
cat("second file", file = file.path(tmp, "mydir", "file2"))

zipfile <- tempfile(fileext = ".zip")
zip::zip(zipfile, "mydir", root = tmp)

## List contents
zip_list(zipfile)

## Extract and inspect result
tmp2 <- tempfile()
result <- unzip(zipfile, exdir = tmp2)
result[, c("filename", "path")]
```

`unzip_process` *Class for an external unzip process*

Description

`unzip_process()` returns an R6 class that represents an unzip process. It is implemented as a subclass of processx::process.

Usage

```
unzip_process()
```

Value

An `unzip_process` R6 class object, a subclass of processx::process, or a subclass of callr::r_process when the fallback is active (see the Fallback section below).

Using the `unzip_process` class

```
up <- unzip_process()$new(zipfile, exdir = ".", poll_connection = TRUE,
                          stderr = tempfile(), ...)
```

See processx::process for the class methods.

Arguments:

- `zipfile`: Path to the zip file to uncompress.
- `exdir`: Directory to uncompress the archive to. If it does not exist, it will be created.
- `poll_connection`: passed to the `initialize` method of processx::process, it allows using `processx::poll()` or the `poll_io()` method to poll for the completion of the process.
- `stderr`: passed to the `initialize` method of processx::process, by default the standard error is written to a temporary file. This file can be used to diagnose errors if the process failed.
- `...` passed to the `initialize` method of processx::process.

Fallback

`unzip_process()` normally runs the bundled `cmdunzip` native executable via processx::process. If the executable cannot be found or fails its self-test it falls back to running `unzip()` in a background R process via callr::r_process. This may happen when system policies do not allow starting the `cmdunzip` executable., The fallback class has the same interface but inherits from callr::r_process instead of processx::process.

Set the environment variable `R_ZIP_PROCESS_FALLBACK=true` to force the fallback unconditionally.

Encoding

The `unzip_process` class does not support the `encoding` argument of `unzip()`. Non-UTF-8 filenames are decoded using the IBM CP437 fallback. Use `unzip()` directly if you need to handle ZIP files with filenames in other encodings (e.g. CP932).

Examples

```
ex <- system.file("example.zip", package = "zip")
tmp <- tempfile()
up <- unzip_process()$new(ex, exdir = tmp)
up$wait()
up$get_exit_status()
dir(tmp)
```

zip *Compress Files into 'zip' Archives*

Description

`zip()` creates a new zip archive file.

Usage

```
zip(
  zipfile,
  files,
  recurse = TRUE,
  compression_level = 9,
  include_directories = TRUE,
  root = ".",
  mode = c("mirror", "cherry-pick"),
  keys = NULL,
  password = NULL,
  encryption = c("aes256", "aes128", "zipcrypto")
)

zipr(
  zipfile,
  files,
  recurse = TRUE,
  compression_level = 9,
  include_directories = TRUE,
  root = ".",
  mode = c("cherry-pick", "mirror"),
  keys = NULL,
  password = NULL,
  encryption = c("aes256", "aes128", "zipcrypto")
)

zip_append(
  zipfile,
  files,
  recurse = TRUE,
  compression_level = 9,
  include_directories = TRUE,
  root = ".",
  mode = c("mirror", "cherry-pick"),
  keys = NULL,
  password = NULL,
  encryption = c("aes256", "aes128", "zipcrypto")
)
```

```
zipr_append(
  zipfile,
  files,
  recurse = TRUE,
  compression_level = 9,
  include_directories = TRUE,
  root = ".",
  mode = c("cherry-pick", "mirror"),
  keys = NULL,
  password = NULL,
  encryption = c("aes256", "aes128", "zipcrypto")
)
```

Arguments

zipfile	The zip file to create. If the file exists, `zip` overwrites it, but `zip_append` appends to it. If it is a directory an error is thrown.
files	Character vector of paths to files to add to the archive. See details below about absolute and relative path names.
recurse	Whether to add the contents of directories recursively.
compression_level	A number between 1 and 9. 9 compresses best, but it also takes the longest.
include_directories	Whether to explicitly include directories in the archive. Including directories might confuse MS Office when reading docx files, so set this to `FALSE` for creating them.
root	Change to this working directory before creating the archive.
mode	Selects how files and directories are stored in the archive. It can be `"mirror"` or `"cherry-pick"`. See "Relative Paths" below for details.
keys	An optional character vector of the same length as `files`, specifying the paths of the corresponding entries inside the zip archive. For a file, the key is the exact archive path. For a directory, the key becomes the directory prefix under which all contents are stored. If `NULL` (default), paths are determined by `mode`. `"."` may not appear in `files` when `keys` is specified.
password	Password for encrypting the archive entries. It can be a string, a raw vector of bytes, or a zero-argument function that returns one of these. If `NULL` (the default), the `zip_password` option is consulted; if that is also `NULL`, entries are stored unencrypted. The password is interpreted as UTF-8 bytes regardless of the current locale, which matches the WinZip/7-Zip convention and ensures interoperability across platforms.
encryption	Encryption scheme to use when `password` is not `NULL`. `"aes256"` (the default) and `"aes128"` use WinZip AES encryption (AES-256 or AES-128 in CTR mode, key derived via PBKDF2-HMAC-SHA1, with an HMAC-SHA1 authentication tag). This scheme is supported by 7-Zip, WinZip, and macOS Archive Utility. `"zipcrypto"` uses the legacy PKWARE ZipCrypto stream cipher, which is

cryptographically weak and should only be used for compatibility with tools that do not support AES encryption.

Details

`zip_append()` appends compressed files to an existing 'zip' file.

Relative paths:

`zip()` and `zip_append()` can run in two different modes: mirror mode and cherry picking mode. They handle the specified `files` differently.

Mirror mode:
Mirror mode is for creating the zip archive of a directory structure, exactly as it is on the disk. The current working directory will be the root of the archive, and the paths will be fully kept. zip changes the current directory to `root` before creating the archive.
E.g. consider the following directory structure:

```
.
|-- foo
|   |-- bar
|   |   |-- file1
|   |   `-- file2
|   `-- bar2
`-- foo2
    `-- file3
```

Assuming the current working directory is `foo`, the following zip entries are created by `zip`:

```
setwd("foo")
zip::zip("../test.zip", c("bar/file1", "bar2", "../foo2"))
#> Warning in warn_for_dotdot(data$key): Some paths reference parent directory,
#> creating non-portable zip file
zip_list("../test.zip")[, "filename", drop = FALSE]
#> # A data frame: 4 x 1
#>   filename
#>   <chr>
#> 1 bar/file1
#> 2 bar2/
#> 3 ../foo2/
#> 4 ../foo2/file3
```

Note that zip refuses to store files with absolute paths, and chops off the leading `/` character from these file names. This is because only relative paths are allowed in zip files.

Cherry picking mode:
In cherry picking mode, the selected files and directories will be at the root of the archive. This mode is handy if you want to select a subset of files and directories, possibly from different paths and put all of them in the archive, at the top level.
Here is an example with the same directory structure as above:

```
zip::zip(
  "../test2.zip",
  c("bar/file1", "bar2", "../foo2"),
  mode = "cherry-pick"
```

```
)
zip_list("../test2.zip")[, "filename", drop = FALSE]
#> # A data frame: 4 x 1
#>   filename
#>   <chr>
#> 1 file1
#> 2 bar2/
#> 3 foo2/
#> 4 foo2/file3
```

From zip version 2.3.0, `"."` has a special meaning in the `files` argument: it will include the files (and possibly directories) within the current working directory, but **not** the working directory itself. Note that this only applies to cherry picking mode.

Permissions::

`zip()` (and `zip_append()`, etc.) add the permissions of the archived files and directories to the ZIP archive, on Unix systems. Most zip and unzip implementations support these, so they will be recovered after extracting the archive.

Note, however that the owner and group (uid and gid) are currently omitted, even on Unix.

`zipr()` **and** `zipr_append()`**:**

These functions exist for historical reasons. They are identical to `zip()` and `zip_append()` with a different default for the `mode` argument.

Value

The name of the created zip file, invisibly.

Examples

```
## Some files to zip up. We will run all this in the R session's
## temporary directory, to avoid messing up the user's workspace.
dir.create(tmp <- tempfile())
dir.create(file.path(tmp, "mydir"))
cat("first file", file = file.path(tmp, "mydir", "file1"))
cat("second file", file = file.path(tmp, "mydir", "file2"))

zipfile <- tempfile(fileext = ".zip")
zip::zip(zipfile, "mydir", root = tmp)

## List contents
zip_list(zipfile)

## Add another file
cat("third file", file = file.path(tmp, "mydir", "file3"))
zip_append(zipfile, file.path("mydir", "file3"), root = tmp)
zip_list(zipfile)
```

`zip_list` *List Files in a 'zip' Archive*

Description

List Files in a 'zip' Archive

Usage

```
zip_list(zipfile, encoding = NULL)
```

Arguments

`zipfile`	Path to an existing ZIP file.
`encoding`	Encoding to use for entry filenames. ZIP files signal UTF-8 filenames via a flag in each entry; those are always decoded as UTF-8 regardless of `encoding`. For entries without that flag, `encoding` is used; `NULL` (the default) falls back to IBM CP437, which is what the ZIP specification prescribes for legacy entries. The value is passed to `iconv()`.

Details

Note that `crc32` is formatted using `as.hexmode()`. `offset` refers to the start of the local zip header for each entry. Following the approach of `seek()` it is stored as a `numeric` rather than an `integer` vector and can therefore represent values up to `2^53-1` (9 PB).

Value

A data frame with columns: `filename`, `compressed_size`, `uncompressed_size`, `timestamp`, `permissions`, `crc32`, `offset`, `type` and `encryption`. `type` is one of `file`, `block_device`, `character_device`, `directory`, `FIFO`, `symlink` or `socket`. `encryption` is one of `none`, `aes128`, `aes192`, `aes256`, `zipcrypto`, or `NA` if encrypted but the scheme cannot be determined.

See Also

Other zip/unzip functions: `unzip()`

`zip_process` *Class for an external zip process*

Description

`zip_process()` returns an R6 class that represents a zip process. It is implemented as a subclass of processx::process.

Usage

```
zip_process()
```

Value

A `zip_process` R6 class object, a subclass of processx::process.

Using the `zip_process` class

```
zp <- zip_process()$new(zipfile, files, recurse = TRUE,
                        poll_connection = TRUE,
                        stderr = tempfile(), ...)
```

See processx::process for the class methods.

Arguments:

- `zipfile`: Path to the zip file to create.
- `files`: Character vector of paths to files to add to the archive. Each specified file or directory in is created as a top-level entry in the zip archive.
- `recurse`: Whether to add the contents of directories recursively.
- `include_directories`: Whether to explicitly include directories in the archive. Including directories might confuse MS Office when reading docx files, so set this to `FALSE` for creating them.
- `poll_connection`: passed to the `initialize` method of processx::process, it allows using `processx::poll()` or the `poll_io()` method to poll for the completion of the process.
- `stderr`: passed to the `initialize` method of processx::process, by default the standard error is written to a temporary file. This file can be used to diagnose errors if the process failed.
- `...` passed to the `initialize` method of processx::process.

Examples

```
dir.create(tmp <- tempfile())
write.table(iris, file = file.path(tmp, "iris.ssv"))
zipfile <- tempfile(fileext = ".zip")
zp <- zip_process()$new(zipfile, tmp)
zp$wait()
zp$get_exit_status()
zip_list(zipfile)
```

Index