Package ‘fs’

April 4, 2020

Title Cross-Platform File System Operations Based on 'libuv'
Version 1.4.1
Description A cross-platform interface to file system operations, built on top of the 'libuv' C library.
Depends R (>= 3.1)
License GPL-3
Encoding UTF-8
LazyData true
ByteCompile true
Imports methods
SystemRequirements GNU make
RoxygenNote 7.0.2
BugReports https://github.com/r-lib/fs/issues
Copyright file COPYRIGHTS
Suggests testthat, covr, pillar (>= 1.0.0), tibble (>= 1.1.0), crayon, rmarkdown, knitr, withr, spelling
VignetteBuilder knitr
Language en-US
NeedsCompilation yes
Author Jim Hester [aut, cre], Hadley Wickham [aut], libuv project contributors [cph] (libuv library), Joyent, Inc. and other Node contributors [cph] (libuv library), RStudio [cph, fnd]
Maintainer Jim Hester <james.f.hestergmail.com>
Repository CRAN
Date/Publication 2020-04-04 00:30:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>2</td>
</tr>
<tr>
<td>create</td>
<td>4</td>
</tr>
<tr>
<td>delete</td>
<td>5</td>
</tr>
<tr>
<td>dir_ls</td>
<td>6</td>
</tr>
<tr>
<td>dir_tree</td>
<td>8</td>
</tr>
<tr>
<td>file_access</td>
<td>8</td>
</tr>
<tr>
<td>file_chmod</td>
<td>9</td>
</tr>
<tr>
<td>file_chown</td>
<td>10</td>
</tr>
<tr>
<td>file_info</td>
<td>11</td>
</tr>
<tr>
<td>file_move</td>
<td>12</td>
</tr>
<tr>
<td>file_show</td>
<td>13</td>
</tr>
<tr>
<td>file_temp</td>
<td>13</td>
</tr>
<tr>
<td>file_touch</td>
<td>14</td>
</tr>
<tr>
<td>fs_bytes</td>
<td>15</td>
</tr>
<tr>
<td>fs_path</td>
<td>16</td>
</tr>
<tr>
<td>fs_perms</td>
<td>16</td>
</tr>
<tr>
<td>id</td>
<td>18</td>
</tr>
<tr>
<td>is_absolute_path</td>
<td>18</td>
</tr>
<tr>
<td>is_file</td>
<td>19</td>
</tr>
<tr>
<td>link_path</td>
<td>20</td>
</tr>
<tr>
<td>path</td>
<td>20</td>
</tr>
<tr>
<td>path_expand</td>
<td>21</td>
</tr>
<tr>
<td>path_file</td>
<td>22</td>
</tr>
<tr>
<td>path_filter</td>
<td>23</td>
</tr>
<tr>
<td>path_math</td>
<td>24</td>
</tr>
<tr>
<td>path_package</td>
<td>26</td>
</tr>
<tr>
<td>path_sanitize</td>
<td>27</td>
</tr>
<tr>
<td>path_tidy</td>
<td>28</td>
</tr>
</tbody>
</table>

Index 29

copy Copy files, directories or links

Description

file_copy() copies files.
link_copy() creates a new link pointing to the same location as the previous link.
dir_copy() copies the directory recursively at the new location.
Usage

```r
file_copy(path, new_path, overwrite = FALSE)
dir_copy(path, new_path, overwrite = FALSE)
link_copy(path, new_path, overwrite = FALSE)
```

Arguments

- `path`: A character vector of one or more paths.
- `new_path`: A character vector of paths to the new locations.
- `overwrite`: Overwrite files if they exist. If this is `FALSE` and the file exists an error will be thrown.

Details

The behavior of `dir_copy()` differs slightly than that of `file.copy()` when `overwrite = TRUE`. The directory will always be copied to `new_path`, even if the name differs from the basename of `path`.

Value

The new path (invisibly).

Examples

```r
file_create("foo")
file_copy("foo", "bar")
try(file_copy("foo", "bar"))
file_copy("foo", "bar", overwrite = TRUE)
file_delete(c("foo", "bar"))

dir_create("foo")
# Create a directory and put a few files in it
files <- file_create(c("foo/bar", "foo/baz"))
file_exists(files)

# Copy the directory
dir_copy("foo", "foo2")
file_exists(path("foo2", path_file(files)))

# Create a link to the directory
link_create(path_abs("foo"), "loo")
link_path("loo")
link_copy("loo", "loo2")
link_path("loo2")

# Cleanup
dir_delete(c("foo", "foo2"))
```
create  Create files, directories, or links

Description

The functions file_create() and dir_create() ensure that path exists; if it already exists it will be left unchanged. That means that compared to file.create(), file_create() will not truncate an existing file, and compared to dir.create(), dir_create() will silently ignore existing directories.

Usage

file_create(path, ..., mode = "u=rw,go=r")

dir_create(path, ..., mode = "u=rwx,go=rx", recurse = TRUE, recursive)

link_create(path, new_path, symbolic = TRUE)

Arguments

path A character vector of one or more paths.
... Additional arguments passed to path()
mode If file/directory is created, what mode should it have?
Links do not have mode; they inherit the mode of the file they link to.
recurse should intermediate directories be created if they do not exist?
recursive (Deprecated) If TRUE recurse fully.
new_path The path where the link should be created.
symbolic Boolean value determining if the link should be a symbolic (the default) or hard link.

Value

The path to the created object (invisibly).

Examples

file_create("foo")
is_file("foo")
# dir_create applied to the same path will fail
try(dir_create("foo"))
dir_create("bar")
delete

```
is_dir("bar")
# file_create applied to the same path will fail
try(file_create("bar"))

# Cleanup
dir_delete("foo")
dir_delete("bar")
```

---

**Description**

file_delete() and link_delete() delete file and links. Compared to file.remove they always fail if they cannot delete the object rather than changing return value or signalling a warning. If any inputs are directories, they are passed to dir_delete(), so file_delete() can therefore be used to delete any filesystem object.

dir_delete() will first delete the contents of the directory, then remove the directory. Compared to unlink it will always throw an error if the directory cannot be deleted rather than being silent or signalling a warning.

**Usage**

```
file_delete(path)
dir_delete(path)
link_delete(path)
```

**Arguments**

- **path** A character vector of one or more paths.

**Value**

The deleted paths (invisibly).

**Examples**

```
# create a directory, with some files and a link to it
dir_create("dir")
files <- file_create(path("dir", letters[1:5]))
link <- link_create(path_abs("dir"), "link")

# All files created
dir_exists("dir")
file_exists(files)
```
# Delete a file
file_delete(files[1])

# The link still exists, but what it points to does not.
link_exists("link")
dir_exists(link_path("link"))

# Delete the link
link_delete("link")
link_exists("link")

---

dir_ls

### List files

dir_ls() is equivalent to the `ls` command. It returns filenames as a named `fs_path` character vector. The names are equivalent to the values, which is useful for passing onto functions like `purrr::map_dfr()`.

dir_info() is equivalent to `ls -l` and a shortcut for `file_info(dir_ls())`.

dir_map() applies a function `fun()` to each entry in the path and returns the result in a list.

dir_walk() calls `fun` for its side-effect and returns the input path.

### Usage

define dir_ls(
    path = ".",
    all = FALSE,
    recurse = FALSE,
    type = "any",
    glob = NULL,
    regexp = NULL,
    invert = FALSE,
    fail = TRUE,
    ...,
    recursive
)
```r

Arguments

path         A character vector of one or more paths.
all          If TRUE hidden files are also returned.
recurse      If TRUE recurse fully, if a positive number the number of levels to recurse.
type         File type(s) to return, one or more of "any", "file", "directory", "symlink", "FIFO",
              "socket", "character_device" or "block_device".
glob         A wildcard aka globbing pattern (e.g. *.csv) passed on to grep() to filter paths.
regexp       A regular expression (e.g.PAL|csv$) passed on to grep() to filter paths.
invert       If TRUE return files which do not match
fail         Should the call fail (the default) or warn if a file cannot be accessed.
...          Additional arguments passed to grep.
recursive    (Deprecated) If TRUE recurse fully.
fun          A function, taking one parameter, the current path entry.

```

Examples

```r
dir_ls(R.home("share"), type = "directory")

# Create a shorter link
link_create(system.file(package = "base"), "base")

dir_ls("base", recurse = TRUE, glob = ".*\R")

dir_map("base", identity)

dir_walk("base", str)

dir_info("base")

# Cleanup
link_delete("base")
```

---

**dir_tree**

*Print contents of directories in a tree-like format*

**Description**

Print contents of directories in a tree-like format

**Usage**

```r
dir_tree(path = ".", recurse = TRUE, ...)
```

**Arguments**

- `path` A path to print the tree from
- `recurse` If TRUE recurse fully, if a positive number the number of levels to recurse.
- `...` Additional arguments passed to `dir_ls`.

---

**file_access**

*Query for existence and access permissions*

**Description**

`file_exists(path)` is a shortcut for `file_access(x,"exists")`; `dir_exists(path)` and `link_exists(path)` are similar but also check that the path is a directory or link, respectively.
Usage

```r
file_access(path, mode = "exists")

file_exists(path)

dir_exists(path)

link_exists(path)
```

**Arguments**

- **path**: A character vector of one or more paths.
- **mode**: A character vector containing one or more of 'exists', 'read', 'write', 'execute'.

**Details**

**Cross-compatibility warning**: There is no executable bit on Windows. Checking a file for mode 'execute' on Windows, e.g. `file_access(x, "execute")` will always return TRUE.

**Value**

A logical vector, with names corresponding to the input path.

**Examples**

```r
file_access("/")
file_access("/", "read")
file_access("/", "write")

file_exists("WOMBATS")
```

---

**file_chmod**

*Change file permissions*

**Description**

Change file permissions

**Usage**

```r
file_chmod(path, mode)
```

**Arguments**

- **path**: A character vector of one or more paths.
- **mode**: A character representation of the mode, in either hexadecimal or symbolic format.
Details

**Cross-compatibility warning:** File permissions differ on Windows from POSIX systems. Windows does not use an executable bit, so attempting to change this will have no effect. Windows also does not have user groups, so only the user permissions (u) are relevant.

Examples

```r
file_create("foo", mode = "000")
file_chmod("foo", "777")
file_info("foo")$permissions

file_chmod("foo", "u-x")
file_info("foo")$permissions

file_chmod("foo", "a-wrx")
file_info("foo")$permissions

file_chmod("foo", "u+wr")
file_info("foo")$permissions

# It is also vectorized
files <- c("foo", file_create("bar", mode = "000"))
file_chmod(files, "a+rwx")
file_info(files)$permissions

file_chmod(files, c("644", "600"))
file_info(files)$permissions
```

---

**file_chown**

Change owner or group of a file

Description

Change owner or group of a file

Usage

```r
file_chown(path, user_id = NULL, group_id = NULL)
```

Arguments

- **path**
  A character vector of one or more paths.

- **user_id**
  The user id of the new owner, specified as a numeric ID or name. The R process must be privileged to change this.

- **group_id**
  The group id of the new owner, specified as a numeric ID or name.
Description

Compared to [file.info] the full results of a stat(2) system call are returned and some columns are returned as S3 classes to make manipulation more natural. On systems which do not support all metadata (such as Windows) default values are used.

Usage

file_info(path, fail = TRUE, follow = FALSE)

file_size(path, fail = TRUE)

Arguments

path A character vector of one or more paths.
fail Should the call fail (the default) or warn if a file cannot be accessed.
follow If TRUE, symbolic links will be followed (recursively) and the results will be that of the final file rather than the link.

Value

A data.frame with metadata for each file. Columns returned are as follows.

path The input path, as a fs_path() character vector.
type The file type, as a factor of file types.
size The file size, as a fs_bytes() numeric vector.
permissions The file permissions, as a fs_perms() integer vector.
modification_time The time of last data modification, as a POSIXct datetime.
user The file owner name - as a character vector.
group The file group name - as a character vector.
device_id The file device id - as a numeric vector.
hard_links The number of hard links to the file - as a numeric vector.
special_device_id The special device id of the file - as a numeric vector.
inode The inode of the file - as a numeric vector.
block_size The optimal block for the file - as a numeric vector.
blocks The number of blocks allocated for the file - as a numeric vector.
flags The user defined flags for the file - as an integer vector.
generation The generation number for the file - as a numeric vector.
access_time  The time of last access - as a POSIXct datetime.
change_time  The time of last file status change - as a POSIXct datetime.
birth_time   The time when the inode was created - as a POSIXct datetime.

See Also

dir_info() to display file information for files in a given directory.

Examples

write.csv(mtcars, "mtcars.csv")
file_info("mtcars.csv")

# Files in the working directory modified more than 20 days ago
files <- file_info(dir_ls())
files$path[difftime(Sys.time(), files$modification_time, units = "days") > 20]

# Cleanup
file_delete("mtcars.csv")

---

file_move  Move or rename files

Description

Compared to file.rename file_move() always fails if it is unable to move a file, rather than signaling a Warning and returning an error code.

Usage

file_move(path, new_path)

Arguments

path        A character vector of one or more paths.
new_path    New file path. If new_path is existing directory, the file will be moved into that directory; otherwise it will be moved/renamed to the full path. Should either be the same length as path, or a single directory.

Value

The new path (invisibly).
Examples

```r
file_create("foo")
file_move("foo", "bar")
file_exists(c("foo", "bar"))
file_delete("bar")
```

---

**file_show**  
Open files or directories

**Description**

Open files or directories

**Usage**

```r
file_show(path = ".", browser = getOption("browser"))
```

**Arguments**

- `path`: A character vector of one or more paths.
- `browser`: A non-empty character string giving the name of the program to be used as the HTML browser. It should be in the PATH, or a full path specified. Alternatively, an R function to be called to invoke the browser.
  Under Windows NULL is also allowed (and is the default), and implies that the file association mechanism will be used.

**Value**

The directories that were opened (invisibly).

---

**file_temp**  
Create names for temporary files

**Description**

`file_temp()` returns the name which can be used as a temporary file.

**Usage**

```r
c > file_temp()
c > file_temp_push("/foo")
c > file_temp_pop()
c > path_temp("/bar")
```
Arguments

- **pattern**: A character vector with the non-random portion of the name.
- **tmp_dir**: The directory the file will be created in.
- **ext**: The file extension of the temporary file.
- **path**: A character vector of one or more paths.
- **...**: Additional paths appended to the temporary directory by `path()`.

Details

- `file_temp_push()` can be used to supply deterministic entries in the temporary file stack. This can be useful for reproducibility in like example documentation and vignettes.
- `file_temp_pop()` can be used to explicitly remove an entry from the internal stack, however generally this is done instead by calling `file_temp()`.
- `path_temp()` constructs a path within the session temporary directory.

Examples

```r
path_temp()
path_temp("does-not-exist")

file_temp()
file_temp(ext = "png")
file_temp("image", ext = "png")

# You can make the temp file paths deterministic
file_temp_push(letters)
file_temp()
file_temp()

# Or explicitly remove values
while (!is.null(file_temp_pop())) next
file_temp_pop()
```

Description

Unlike the touch POSIX utility this does not create the file if it does not exist. Use `file_create()` to do this if needed.

Usage

```r
file_touch(path, access_time = Sys.time(), modification_time = access_time)
```
Arguments

- **path**: A character vector of one or more paths.
- **access_time, modification_time**: The times to set, inputs will be coerced to POSIXct objects.

Examples

```r
file_create("foo")
file_touch("foo", "2018-01-01")
file_info("foo")[[c("access_time", "modification_time", "change_time", "birth_time")]]
```

---

### fs_bytes

**Human readable file sizes**

**Description**

Construct, manipulate and display vectors of file sizes. These are numeric vectors, so you can compare them numerically, but they can also be compared to human readable values such as ’10MB’.

**Usage**

```r
as_fs_bytes(x)
fs_bytes(x)
```

**Arguments**

- **x**: A numeric or character vector. Character representations can use shorthand sizes (see examples).

**Examples**

```r
fs_bytes("1")
fs_bytes("1K")
fs_bytes("1Kb")
fs_bytes("1Kib")
fs_bytes("1MB")

fs_bytes("1KB") < "1MB"

sum(fs_bytes(c("1MB", "5MB", "500KB")))
```
fs_perms

Description

fs_perms() objects help one create and modify file permissions easily. They support both numeric input, octal and symbolic character representations. Compared to octmode they support symbolic representations and display the mode the same format as ls on POSIX systems.

Usage

as_fs_perms(x, ...)

fs_perms(x, ...)

Arguments

x An object which is to be coerced to a fs_perms object. Can be an number or octal character representation, including symbolic representations.

... Additional arguments passed to methods.
Details

On POSIX systems the permissions are displayed as a 9 character string with three sets of three characters. Each set corresponds to the permissions for the user, the group and other (or default) users.

If the first character of each set is a "r", the file is readable for those users, if a ".", it is not readable.

If the second character of each set is a "w", the file is writable for those users, if a "-", it is not writable.

The third character is more complex, and is the first of the following characters which apply.

- 'S' If the character is part of the owner permissions and the file is not executable or the directory is not searchable by the owner, and the set-user-id bit is set.
- 'S' If the character is part of the group permissions and the file is not executable or the directory is not searchable by the group, and the set-group-id bit is set.
- 'T' If the character is part of the other permissions and the file is not executable or the directory is not searchable by others, and the 'sticky' (S_ISVTX) bit is set.
- 's' If the character is part of the owner permissions and the file is executable or the directory searchable by the owner, and the set-user-id bit is set.
- 's' If the character is part of the group permissions and the file is executable or the directory searchable by the group, and the set-group-id bit is set.
- 't' If the character is part of the other permissions and the file is executable or the directory searchable by others, and the "sticky" (S_ISVTX) bit is set.
- 'x' The file is executable or the directory is searchable.
- '-' If none of the above apply. Most commonly the third character is either 'x' or -.

On Windows the permissions are displayed as a 3 character string where the third character is only - or x.

Examples

# Integer and numeric
fs_perms(420L)
fs_perms(c(511, 420))

# Octal
fs_perms("777")
fs_perms(c("777", "644"))

# Symbolic
fs_perms("a+rwx")
fs_perms(c("a+rwx", "u+rw,go+r"))

# Use the \\& and \\| operators to check for certain permissions
(fs_perms("777") & "u+r") == "u+r"
id

Lookup Users and Groups on a system

Description
These functions use the GETPWENT(3) and GETGREN(3) system calls to query users and groups respectively.

Usage

group_ids()
user_ids()

Value
They return their results in a data.frame. On Windows both functions return an empty data.frame because Windows does not have user or group ids.

Examples

# list first 6 groups
head(group_ids())

# list first 6 users
head(user_ids())

is_absolute_path

Test if a path is an absolute path

Description
Test if a path is an absolute path

Usage

is_absolute_path(path)

Arguments

path A character vector of one or more paths.

Examples

is_absolute_path("/foo")
is_absolute_path("C:\foo")
is_absolute_path("\\\myserver\\foo\\bar")
is_absolute_path("foo/bar")
is_file

Functions to test for file types

Description

Functions to test for file types

Usage

is_file(path, follow = TRUE)

is_dir(path, follow = TRUE)

is_link(path)

is_file_empty(path, follow = TRUE)

Arguments

path
A character vector of one or more paths.

follow
If TRUE, symbolic links will be followed (recursively) and the results will be that of the final file rather than the link.

Value

A named logical vector, where the names give the paths. If the given object does not exist, NA is returned.

See Also

file_exists(), dir_exists() and link_exists() if you want to ensure that the path also exists.

Examples

dir_create("d")

file_create("d/file.txt")
dir_create("d/dir")
link_create(path(path_abs("d"), "file.txt"), "d/link")

paths <- dir_ls("d")
is_file(paths)
is_dir(paths)
is_link(paths)

# Cleanup
dir_delete("d")
link_path

Read the value of a symbolic link

Description
Read the value of a symbolic link

Usage
link_path(path)

Arguments
path                       A character vector of one or more paths.

Value
A tidy path to the object the link points to.

Examples
file_create("foo")
link_create(path_abs("foo"), "bar")
link_path("bar")

# Cleanup
file_delete(c("foo", "bar"))

path

Construct path to a file or directory

Description
path() constructs a relative path, path_wd() constructs an absolute path from the current working directory.

Usage
path(..., ext = "")
path_wd(..., ext = ")
Arguments

... character vectors, if any values are NA, the result will also be NA. The paths follow the recycling rules used in the tibble package, namely that only length 1 arguments are recycled.

ext An optional extension to append to the generated path.

See Also

path_home(), path_package() for functions to construct paths relative to the home and package directories respectively.

Examples

path("foo", "bar", "baz", ext = "zip")

path("foo", letters[1:3], ext = "txt")

Description

• path_expand() performs tilde expansion on a path, replacing instances of ~ or ~user with the user’s home directory.

• path_home() constructs a path within the expanded users home directory, calling it with no arguments can be useful to verify what fs considers the home directory.

• path_expand_r() and path_home_r() are equivalents which always use R’s definition of the home directory.

Usage

path_expand(path)

path_expand_r(path)

path_home(...)

path_home_r(...)

Arguments

path A character vector of one or more paths.

... Additional paths appended to the home directory by path().
Details

`path.expand()` differs from `base::path.expand()` in the interpretation of the home directory of Windows. In particular `path.expand()` uses the path set in the `USERPROFILE` environment variable and, if unset, then uses `HOMEDRIVE/HOMEPATH`.

In contrast `base::path.expand()` first checks for `R_USER` then `HOME`, which in the default configuration of R on Windows are both set to the user’s document directory, e.g. C:\Users\username\Documents. `base::path.expand()` also does not support `~otheruser` syntax on Windows, whereas `path.expand()` does support this syntax on all systems.

This definition makes fs more consistent with the definition of home directory used on Windows in other languages, such as python and rust. This is also more compatible with external tools such as git and ssh, both of which put user-level files in `USERPROFILE` by default. It also allows you to write portable paths, such as `~/Desktop` that points to the Desktop location on Windows, macOS and (most) Linux systems.

Users can set the `R_FS_HOME` environment variable to override the definitions on any platform.

See Also

R for Windows FAQ - 2.14 for behavior of `base::path.expand()`.

Examples

```r
# Expand a path
path.expand("~/bin")

# You can use `path_home()` without arguments to see what is being used as
# the home directory.
path_home()
path_home("R")

# This will likely differ from the above on Windows
path_home_r()
```

path_file

Manipulate file paths

Description

`path_file()` returns the filename portion of the path, `path_dir()` returns the directory portion. `path_ext()` returns the last extension (if any) for a path. `path_ext_remove()` removes the last extension and returns the rest of the path. `path_ext_set()` replaces the extension with a new extension. If there is no existing extension the new extension is appended.
Usage

path_file(path)
path_dir(path)
path_ext(path)
path_ext_remove(path)
path_ext_set(path, ext)
path_ext(path) <- value

Arguments

path A character vector of one or more paths.
ext, value The new file extension.

Details

Note because these are not full file paths they return regular character vectors, not fs_path() objects.

See Also

base::basename(), base::dirname()

Examples

path_file("dir/file.zip")
path_dir("dir/file.zip")
path_ext("dir/file.zip")
path_ext("file.tar.gz")
path_ext_remove("file.tar.gz")

# Only one level of extension is removed
path_ext_set(path_ext_remove("file.tar.gz"), "zip")

path_filter      Filter paths

Description

Filter paths
Usage

```r
path_filter(path, glob = NULL, regexp = NULL, invert = FALSE, ...)
```

Arguments

- **path**: A character vector of one or more paths.
- **glob**: A wildcard aka globbing pattern (e.g. `*.csv`) passed on to `grep()` to filter paths.
- **regexp**: A regular expression (e.g. `[]csv$`) passed on to `grep()` to filter paths.
- **invert**: If `TRUE` return files which do *not* match
- **...**: Additional arguments passed to `grep`.

Examples

```r
path_filter(c("foo", "boo", "bar"), glob = "*oo")
path_filter(c("foo", "boo", "bar"), glob = "*oo", invert = TRUE)
path_filter(c("foo", "boo", "bar"), regexp = "b.r")
```

---

**path_math**

*Path computations*

Description

All functions apart from `path_real()` are purely path computations, so the files in question do not need to exist on the filesystem.

Usage

- `path_real(path)`
- `path_split(path)`
- `path_join(parts)`
- `path_abs(path, start = ".")`
- `path_norm(path)`
- `path_rel(path, start = ".")`
- `path_common(path)`
- `path_has_parent(path, parent)`
Arguments

- **path**: A character vector of one or more paths.
- **parts**: A character vector or a list of character vectors, corresponding to split paths.
- **start**: A starting directory to compute the path relative to.
- **parent**: The parent path.

Value

The new path(s) in an `fs_path` object, which is a character vector that also has class `fs_path`. Except `path_split()`, which returns a list of character vectors of path components.

Functions

- `path_real`: returns the canonical path, eliminating any symbolic links and the special references `~`, `~user`, `.`, and `..`, i.e., it calls `path_expand()` (literally) and `path_norm()` (effectively).
- `path_split`: splits paths into parts.
- `path_join`: joins parts together. The inverse of `path_split()`. See `path()` to concatenate vectorized strings into a path.
- `path_abs`: returns a normalized, absolute version of a path.
- `path_norm`: eliminates . references and rationalizes up-level .. references, so A/./B and A/foo/../B both become A/B, but ../B is not changed. If one of the paths is a symbolic link, this may change the meaning of the path, so consider using `path_real()` instead.
- `path_rel`: computes the path relative to the `start` path, which can be either an absolute or relative path.
- `path_common`: finds the common parts of two (or more) paths.
- `path_has_parent`: determine if a path has a given parent.

See Also

- `path_expand()` for expansion of user’s home directory.

Examples

```r
dir_create("a")
file_create("a/b")
link_create(path_abs("a"), "c")

# Realize the path
path_real("c/b")

# Split a path
parts <- path_split("a/b")
parts

# Join it together
```


path_join(parts)

# Find the absolute path
path_abs(“..”)

# Normalize a path
path_norm(“a/..b\c/.”)

# Compute a relative path
path_rel(“/foo/abc”, ”/foo/bar/baz”)

# Find the common path between multiple paths
path_common(c(“/foo/bar/baz”, ”/foo/bar/abc”, ”/foo/xyz/123”))

# Cleanup
dir_delete(“a”)
link_delete(“c”)

---

path_package

*Construct a path to a location within an installed or development package*

**Description**

path_package differs from system.file() in that it always returns an error if the package does not exist. It also returns a different error if the file within the package does not exist.

**Usage**

path_package(package, ...)

**Arguments**

package Name of the package to in which to search

... Additional paths appended to the package path by path().

**Details**

path_package() also automatically works with packages loaded with devtools even if the path_package() call comes from a different package.

**Examples**

path_package(“base”)
path_package(“stats”)
path_package(“base”, “INDEX”)
path_package(“splines”, “help”, “AnIndex”)
path sanitize

Sanitize a filename by removing directory paths and invalid characters

Description

path sanitize() removes the following:

- Control characters
- Reserved characters
- Unix reserved filenames (..)
- Trailing periods and spaces (invalid on Windows)
- Windows reserved filenames (CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9) The resulting string is then truncated to 255 bytes in length

Usage

path sanitize(filename, replacement = "")

Arguments

filename A character vector to be sanitized.
replacement A character vector used to replace invalid characters.

See Also

https://www.npmjs.com/package/sanitize-filename, upon which this function is based.

Examples

# potentially unsafe string
str <- '~/\u0001ssh/authorized_keys'
path sanitize(str)

path sanitize("..")
**Description**

Untidy paths are all different, tidy paths are all the same. Tidy paths always use `/` to delimit directories, never have multiple `/` or trailing `/` and have colourised output based on the file type.

**Usage**

`path_tidy(path)`

**Arguments**

`path` A character vector of one or more paths.

**Value**

An `fs_path` object, which is a character vector that also has class `fs_path`
Index

as_fs_bytes (fs_bytes), 15
as_fs_path (fs_path), 16
as_fs_perms (fs_perms), 16
base::basename(), 23
base::dirname(), 23
base::path.expand(), 22
copy, 2
create, 4
delete, 5
dir.create(), 4
dir_copy (copy), 2
dir_create (create), 4
dir_delete (delete), 5
dir_exists (file_access), 8
dir_exists(), 19
dir_info (dir_ls), 6
dir_info(), 12
dir_ls, 6, 8
dir_map (dir_ls), 6
dir_tree, 8
dir_walk (dir_ls), 6
file.create(), 4
file.remove, 5
file.rename, 12
file_access, 8
file_chmod, 9
file_chown, 10
file_copy (copy), 2
file_create (create), 4
file_create(), 14
file_delete (delete), 5
file_exists (file_access), 8
file_exists(), 19
file_info, 11
file_move, 12
file_show, 13
file_size (file_info), 11
file_temp, 13
file_temp_pop (file_temp), 13
file_temp_push (file_temp), 13
file_touch, 14
fs_bytes, 15
fs_bytes(), 11
fs_path, 16
fs_path(), 11
fs_perms, 16
fs_perms(), 11
grep, 7, 24
grep(), 7, 24
group_ids (id), 18
id, 18
is_absolute_path, 18
is_dir (is_file), 19
is_file, 19
is_file_empty (is_file), 19
is_link (is_file), 19
link_copy (copy), 2
link_create (create), 4
link_delete (delete), 5
link_exists (file_access), 8
link_exists(), 19
link_path, 20
octmode, 16
path, 20
path(), 4, 21, 25, 26
path_abs (path_math), 24
path_common (path_math), 24
path_dir (path_file), 22
path_expand, 21
path_expand(), 25
path_expand_r (path_expand), 21
path_ext (path_file), 22
path_ext<- (path_file), 22
path_ext_remove (path_file), 22
path_ext_set (path_file), 22
path_file, 22
path_filter, 23
path_has_parent (path_math), 24
path_home (path_expand), 21
path_home(), 21
path_home_r (path_expand), 21
path_join (path_math), 24
path_math, 24
path_norm (path_math), 24
path_package, 26
path_package(), 21
path_real (path_math), 24
path_rel (path_math), 24
pathsanitize, 27
path_split (path_math), 24
path_split(), 25
path_temp (file_temp), 13
path_tidy, 28
path_wd (path), 20
POSIXct, 11, 12, 15
purrr::map_dfr(), 6

system.file(), 26
unlink, 5
user_ids (id), 18