Package: osfr (via r-universe)

November 1, 2024

Title Interface to the 'Open Science Framework' ('OSF')

Version 0.2.9

Description An interface for interacting with 'OSF' (<https://osf.io>). 'osfr' enables you to access open research materials and data, or create and manage your own private or public projects.

Depends R (>= 3.1.0)

Imports crul (>= 0.7.4), jsonlite, stringi, purrr, tibble (>= 3.0.0), rlang, fs (>= 1.3.0), memoise, httr

License MIT + file LICENSE

URL https://docs.ropensci.org/osfr/, https://github.com/ropensci/osfr

BugReports https://github.com/ropensci/osfr/issues

Suggests dplyr, logger, rprojroot, brio, testthat, knitr, rmarkdown, lintr (>= 2.0), covr, spelling, vcr (>= 0.5)

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

VignetteBuilder knitr

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/osfr

RemoteRef master

RemoteSha 67315ad3cff5d153e21d462f554f877ff8d5bcd3

Contents

`osfr-package` *osfr: R interface to OSF*

Description

osfr provides a suite of functions for interacting with the Open Science Framework (OSF; https://osf.io/).

What is OSF?

OSF is a free and open source project management repository designed to support researchers across their entire project lifecycle. The service includes free cloud storage and file version history, providing a centralized location for all your research materials that can be kept private, shared with select collaborators, or made publicly available with citable DOIs.

Most work on OSF is organized around **projects**. Projects can contain *files*, groups of files in *directories*, and/or files in sub-projects called **components**. Note there is no storage limit on the size of projects but individual files must be < 5Gb.

Resources

- To learn more about OSF check out the helpful series of guides published by the Center for Open Science: https://help.osf.io
- See the vignette for an overview of osfr's features: `vignette("getting_started", package = "osfr")`

Author(s)

Maintainer: Aaron Wolen <aaron@wolen.com> (ORCID)

Authors:

- Chris Hartgerink <chjh@protonmail.com> (ORCID)

Other contributors:

- Timothy York <timothypyork@gmail.com> (ORCID) [contributor]
- Ryan Hafen <rhafen@purdue.edu> [contributor]
- Brian Richards <brian.g.richards@gmail.com> [contributor]
- Courtney Soderberg <courtney@cos.io> (ORCID) [contributor]
- Carl Boettiger <cboettig@gmail.com> (ORCID) [reviewer]
- Heidi Seibold <heidi.seibold@stat.uni-muenchen.de> [reviewer]

See Also

Useful links:

- https://docs.ropensci.org/osfr/
- https://github.com/ropensci/osfr
- Report bugs at https://github.com/ropensci/osfr/issues

`as_id` *Extract OSF identifiers*

Description

Extract OSF GUIDs and Waterbutler IDs from various types of inputs. Valid looking IDs are returned as `osf_id` objects.

Usage

```
as_id(x)
```

Arguments

`x`	An `osf_tbl`, OSF URL, or a generic string containing a GUID or Waterbutler ID.

Value

A character vector with class `osf_id`.

Identifier types

There are 2 types of identifiers you'll encounter on OSF. The first is the globally unique identifier, or GUID, that OSF assigns to every entity. A valid OSF GUID consists of 5 alphanumeric characters. The second type of identifier is specific to files stored on OSF. All file operations on OSF are handled via Waterbutler. A valid Waterbutler ID consists of 24 alphanumeric characters.

Examples

```
## Not run:
# extract a GUID from an OSF URL
proj_id <- as_id("https://osf.io/7zqxp/")

# extract waterbutler IDs from an `osf_tbl_file`` with multiple files
osf_retrieve_node(proj_id) %>%
  osf_ls_files() %>%
  as_id()

## End(Not run)
```

`osf_auth` *Authenticate osfr with a personal access token*

Description

Authorize osfr to interact with your OSF data and OSF account by passing a personal access token (PAT) to `osf_auth()`. If no `token` is provided, `osf_auth()` will attempt to obtain a PAT from the `OSF_PAT` environment variable. However, since osfr checks for the presence of `OSF_PAT` on start-up, this is only necessary if the variable was created or redefined in the middle of a session. See below for additional details and instructions for generating and utilizing your PAT.

Usage

```
osf_auth(token = NULL)
```

Arguments

`token`	OSF personal access token.

Details

Out of the box osfr can only access publicly available projects, components, and files on OSF. In order for osfr to view and manage your private resources you must provide a Personal Access Token (PAT). The following instructions will walk you through the process of generating a PAT and using it to authenticate osfr.

Value

Invisibly returns your OSF PAT along with a message indicating it was registered.

Creating an OSF personal access token

- Navigate to https://osf.io/settings/tokens/
- Click the *New token* button and provide a descriptive name
- Select the scopes (i.e., permissions) you'd like to grant osfr

- Click the *Create* button to generate your PAT
- If successful, your 70 character token will be displayed along with several important warnings you should definitely read over carefully
- You read those warnings, right?
- Copy your token and keep it in a safe place

Using your PAT with osfr

There are two possible approaches for authenticating osfr with your PAT.

The simpler approach is to call the `osf_auth()` function at the start of every new R session and manually paste in your token. Note that your PAT should be treated like a password and, as such, should not be hardcoded into your script.

A better approach is to store your PAT as an environment variable called `OSF_PAT`. Doing so will allow osfr to detect and utilize the token automatically without any need to manually call `osf_auth()`. One way to accomplish this is by creating an `.Renviron` file in your home or working directory that defines the `OSF_PAT` variable. For example:

```
OSF_PAT=bdEEFMCuBtaBoSK11YzyjOdjUjKtWIj2FWxHl6kTBRax7uaeyBghALumTO1kT8RA
```

For new users we suggest adding the `.Renviron` to your home directory so it is automatically applied to all your projects. To verify this was done correctly, restart R and run `Sys.getenv("OSF_PAT")`, which should return your PAT.

References

1. Colin Gillespie and Robin Lovelace (2017). *Efficient R programming*. O'Reilly Press. https://csgillespie.github.io/efficientR/.

Examples

```
## Not run:
# manually authenticate with a PAT
osf_auth("bdEEFMCuBtaBoSK11YzyjOdjUjKtWIj2FWxHl6kTBRax7uaeyBghALumTO1kT8RA")

## End(Not run)
```

---

osf_cp *Copy a file or directory*

Description

Use `osf_cp()` to make a copy of a file or directory in a new location.

Usage

```
osf_cp(x, to, overwrite = FALSE, verbose = FALSE)
```

Arguments

x	An osf_tbl_file containing a single file or directory.
to	Destination where the file or directory will be copied. This can be one of the following: • An osf_tbl_node with a single project or component. • An osf_tbl_file with a single directory.
overwrite	Logical, if a file or directory with the same name already exists at the destination should it be replaced with x?
verbose	Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).

Details

Note that a file (or directory) cannot be moved or copied onto itself, even if overwrite = TRUE.

Value

An osf_tbl_file containing the updated OSF file.

See Also

Other OSF file operations: osf_mkdir(), osf_mv(), osf_rm()

Examples

```
## Not run:
project <- osf_create_project("Flower Data")

write.csv(iris, file = "iris.csv")
data_file <- osf_upload(project,"iris.csv")

# Create a new directory to copy our file to
data_dir <- osf_mkdir(project, "data")

# Copy the file to our data directory
data_file <- osf_cp(data_file, to = data_dir)

# Copy directory to new component
data_comp <- osf_create_component(project, title = "data", category = "data")
data_dir %>%
  osf_cp(to = data_comp) %>%
  osf_open()

## End(Not run)
```

`osf_create` *Create a new project or component on OSF*

Description

Use `osf_create_project()` to create a new top-level project on OSF. A nested component can be created by providing an `osf_tbl_node` containing an existing project or component to `osf_create_component()`'s `x` argument.

Usage

```
osf_create_project(
  title,
  description = NULL,
  public = FALSE,
  category = "project"
)

osf_create_component(
  x,
  title,
  description = NULL,
  public = FALSE,
  category = NULL
)
```

Arguments

`title, description`
: Set a title (required) and, optionally, a description.

`public`
: Logical, should it be publicly available (`TRUE`) or private (`FALSE`, the default)?

`category`
: Character string, specify a category to change the icon displayed on OSF. The defaults are `"project"` for projects and `"uncategorized"` for components. The specified category can be easily changed later on OSF. Valid category options include:

  - analysis
  - communication
  - data
  - hypothesis
  - instrumentation
  - methods and measures
  - procedure
  - project
  - software
  - other

x — An osf_tbl_node with a single OSF project or component that will serve as the new sub-component's parent node.

Value

An osf_tbl_node containing the new project or component.

OSF nodes

Projects and components are both implemented as *nodes* on OSF. The only distinction between the two is that a project is a top-level node, and a component must have a parent node (i.e., must be a sub-component of another project or component). Because projects and components are functionally identical, osfr uses the same osf_tbl_node class to represent both.

References

1. OSF Guides: Create a Project. https://help.osf.io/article/383-creating-a-project.
2. OSF Guides: Create a Component. https://help.osf.io/article/253-create-components.

Examples

```
## Not run:
# create a new public project
project <- osf_create_project(title = "Private OSF Project", public = TRUE)

# add a private component to the new project
component <- osf_create_component(project, title = "Project Data")

## End(Not run)
```

---

osf_download — *Download files and directories from OSF*

Description

Files stored on OSF can be downloaded locally by passing an osf_tbl_file that contains the files and folders of interest. Use path to specify *where* the files should be downloaded, otherwise they are downloaded to your working directory by default.

Usage

```
osf_download(
  x,
  path = NULL,
  recurse = FALSE,
  conflicts = "error",
  verbose = FALSE,
  progress = FALSE
)
```

Arguments

x	An osf_tbl_file containing a single file or directory.
path	Path pointing to a local directory where the downloaded files will be saved. Default is to use the current working directory.
recurse	Applies only to OSF directories. If TRUE, a directory is fully recursed and all nested files and subdirectories are downloaded. Alternatively, a positive number will determine the number of levels to recurse.
conflicts	This determines what happens when a file with the same name exists at the specified destination. Can be one of the following: • "error" (the default): throw an error and abort the file transfer operation. • "skip": skip the conflicting file(s) and continue transferring the remaining files. • "overwrite": replace the existing file with the transferred copy.
verbose	Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).
progress	Logical, if TRUE progress bars are displayed for each file transfer. Mainly useful for transferring large files. For tracking lots of small files, setting verbose = TRUE is more informative.

Value

The same osf_tbl_file passed to x with a new column, "local_path", containing paths to the local files.

Implementation details

Directories are always downloaded from OSF as zip files that contain its entire contents. The logic for handling conflicts and recursion is implemented locally, acting on these files in a temporary location and copying them to path as needed. This creates a *gotcha* if you're downloading directories with large files and assuming that setting conflicts = "skip" and/or limiting recursion will reduce the number of files you're downloading. In such a case, a better strategy would be to use osf_ls_files() to list the contents of the directory and pass that output to osf_download().

A note about synchronization

While osf_download() and osf_upload() allow you to conveniently shuttle files back and forth between OSF and your local machine, it's important to note that **they are not file synchronization functions**. In contrast to something like rsync, osf_download()/osf_upload() do not take into account a file's contents or modification time. Whether you're uploading or downloading, if conflicts = "overwrite", osfr will overwrite the existing file regardless of whether it is the more recent copy. You have been warned.

See Also

- osf_upload() for uploading files to OSF.
- osf_ls_files() for listing files and directories on OSF.

Examples

```
## Not run:
# download a single file
analysis_plan <- osf_retrieve_file("2ryha") %>%
  osf_download()

# verify the file was downloaded locally
file.exists(analysis_plan$local_path)

## End(Not run)
```

`osf_ls_files` *List files and directories on OSF*

Description

List the files and directories in the top-level of an OSF project, component, or directory. Specify a `path` to list the contents of a particular subdirectory.

Usage

```
osf_ls_files(
  x,
  path = NULL,
  type = "any",
  pattern = NULL,
  n_max = 10,
  verbose = FALSE
)
```

Arguments

`x`	One of the following: • An `osf_tbl_node` with a single project or component. • An `osf_tbl_file` with a single directory.
`path`	List files within the specified subdirectory path.
`type`	Filter query by type. Set to `"file"` to list only files, or `"folder"`to list only folders
`pattern`	Character string used to filter for results that contain the substring `"pattern"` in their name. *Note:* this is a fixed, case-insensitive search.
`n_max`	Maximum number of results to return from OSF (default is 10). Set to `Inf` to return *all* results.
`verbose`	Logical, indicating whether to print informative messages about interactions with the OSF API (default `FALSE`).

Value

An osf_tbl_file with one row for each file or directory, ordered by modification time.

See Also

osf_ls_nodes() to generate a list of projects and components.

Examples

```
## Not run:
# Retrieve the Psychology Reproducibility Project from OSF
psych_rp <- osf_retrieve_node("ezum7")

# List all files and directories
osf_ls_files(psych_rp)

# ...only the directories
osf_ls_files(psych_rp, type = "folder")

# ...only PDF files
osf_ls_files(psych_rp, type = "file", pattern = "pdf")

# List the contents of the first directory
osf_ls_files(psych_rp, path = "RPP_SI_Figures")

## End(Not run)
```

---

osf_ls_nodes *List projects or components on OSF*

Description

List the projects or components associated with a user or contained in the top-level of another OSF project or component.

Usage

```
osf_ls_nodes(x, pattern = NULL, n_max = 10, verbose = FALSE)
```

Arguments

x	one of the following: • An osf_tbl_node with a single project or component. • An osf_tbl_user with a single OSF user.
pattern	Character string used to filter for results that contain the substring "pattern" in their name. *Note:* this is a fixed, case-insensitive search.
n_max	Maximum number of results to return from OSF (default is 10). Set to Inf to return *all* results.

verbose — Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).

Value

An osf_tbl_node with one row for each OSF project or component, ordered by modification time.

See Also

osf_ls_files() to generate a list of files and files.

Examples

```
## Not run:
# List your recent projects and components
osf_retrieve_user("me") %>%
  osf_ls_nodes()

# List the first 10 components in the #ScanAllFish project
fish_ctscans <- osf_retrieve_node("ecmz4")
osf_ls_nodes(fish_ctscans)

# Now just the components with scans of species from the Sphyrna genus
osf_ls_nodes(fish_ctscans, pattern = "Sphyrna")

## End(Not run)
```

---

osf_mkdir *Create directories on OSF*

Description

Use osf_mkdir() to add new directories to projects, components, or nested within existing OSF directories. If path contains multiple directory levels (e.g., "data/rawdata") the intermediate-level directories are created automatically. If the directory you're attempting to create already exists on OSF it will be silently ignored and included in the output.

Usage

```
osf_mkdir(x, path, verbose = FALSE)
```

Arguments

x — One of the following:

- An osf_tbl_node with a single OSF project or component.
- An osf_tbl_file containing a single directory.

path — Name of the new directory or a path ending with the new directory.

verbose — Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).

Value

An osf_tbl_file with one row containing the leaf directory specified in path.

See Also

Other OSF file operations: osf_cp(), osf_mv(), osf_rm()

Examples

```
## Not run:
proj <- osf_create_project("Directory Example")

# add directory to the top-level of the Directory Example project
data_dir <- osf_mkdir(proj, path = "data")

# add a subdirectory nested within data/
osf_mkdir(data_dir, path = "rawdata")

# recursively create multiple directory levels within data/
osf_mkdir(data_dir, path = "samples/pcr/qc")

## End(Not run)
```

---

osf_mv *Move a file or directory*

Description

Use osf_mv() to move a file or directory to a new project, component, or subdirectory.

Usage

```
osf_mv(x, to, overwrite = FALSE, verbose = FALSE)
```

Arguments

x	An osf_tbl_file containing a single file or directory.
to	Destination where the file or directory will be moved. This can be one of the following: • An osf_tbl_node with a single project or component. • An osf_tbl_file with a single directory.
overwrite	Logical, if a file or directory with the same name already exists at the destination should it be replaced with x?
verbose	Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).

Details

Note that a file (or directory) cannot be moved or copied onto itself, even if `overwrite = TRUE`.

Value

An `osf_tbl_file` containing the updated OSF file.

See Also

Other OSF file operations: `osf_cp()`, `osf_mkdir()`, `osf_rm()`

Examples

```
## Not run:
# Create an example file to upload to our example project
project <- osf_create_project("Flower Data")

write.csv(iris, file = "iris.csv")
data_file <- osf_upload(project,"iris.csv")

# Create a new directory to move our file to
data_dir <- osf_mkdir(project, "data")

# Move the file to our data directory
data_file <- osf_mv(data_file, to = data_dir)

# Move our data directory to a new component
data_comp <- osf_create_component(project, title = "data", category = "data")
data_dir %>%
  osf_mv(to = data_comp) %>%
  osf_open()

## End(Not run)
```

`osf_open` *Open on OSF*

Description

View a project, component, file, or user profile on OSF with your default web browser.

Usage

```
osf_open(x)
```

Arguments

x	one of the following: • an OSF URL, or a generic string containing a GUID or Waterbutler ID. • an osf_tbl_node with a single project or component. • an osf_tbl_file with a single file or directory. • an osf_tbl_user with a single OSF user.

Examples

```
## Not run:
# Navigate to a project based on its GUID
osf_open("e81xl")

# You can also provide an osf_tbl subclass
crp_file <- osf_retrieve_file("ucpye")
osf_open(crp_file)

## End(Not run)
```

---

osf_refresh *Refresh an OSF entity*

Description

Use osf_refresh() to update one or more entities in an osf_tbl() with the latest information from OSF.

Usage

```
osf_refresh(x)
```

Arguments

x	an osf_tbl.

---

osf_retrieve *Retrieve an entity from OSF*

Description

Create an osf_tbl representation of an existing OSF project, component, file, or user based on the associated unique identifier. Usually this is a 5-character global unique identifier (GUID) but for files or directories, it could also be an 11-character Waterbutler ID. See below for details.

Usage

```
osf_retrieve_user(id)

osf_retrieve_node(id)

osf_retrieve_file(id)
```

Arguments

id	An OSF identifier corresponding to an OSF user, project, component, or file. Set id = "me" to retrieve your own OSF profile.

Value

An osf_tbl_user, osf_tbl_node, or osf_tbl_file containing the corresponding OSF entity.

OSF identifiers

A 5-character GUID is assigned to every user, project, component, and file on OSF and forms the basis for the service's URL scheme. For example the GUID for a project accessible at https://osf.io/ezum7 is simply ezum7. You can learn more about GUIDs in the OSF FAQ.

An important detail is that files and directories are handled internally on OSF by another serviced called Waterbutler, which uses 11-character identifiers. Although Waterbutler IDs are largely hidden from users on https://osf.io, they represent the primary method for identifying files/directories by the OSF API. In fact, files do not receive a GUID until it is viewed directly on https://osf.io and directories never receive a GUID. Therefore, osfr relies on Waterbutler IDs for files and directories, and always includes them (rather than GUIDs) in osf_tbl_file objects.

Retrieving OSF objects

To begin using osfr to interact with resources on OSF you must use one of the following *retrieve* functions to create an osf_tbl that contains the entity of interest. Note the functions are entity-type specific, use:

- osf_retrieve_node() to retrieve a project or component
- osf_retrieve_file() to retrieve a file or directory
- osf_retrieve_user() to retrieve a user

A note on 3rd-party storage providers

While OSF supports integration with a variety of 3rd-party cloud storage providers, osfr can currently only access files stored on the default OSF storage service. Support for additional storage providers is planned for a future release.

Examples

```
## Not run:
 # retrieve your own OSF user profile (must be authenticated, ?osf_auth)
 osf_retrieve_user("me")

# retrieve the Psychology Reproducibility Project (P:RP, osf.io/ezum7)
osf_retrieve_node("ezum7")

# get the first figure from the P:RP
osf_retrieve_file("https://osf.io/7js8c")

## End(Not run)
```

`osf_rm` *Delete an entity from OSF*

Description

Use `osf_rm()` to **permanently** delete a project, component, file or directory from OSF, including any uploaded files, wiki content, or comments contained therein. Because this process is **irreversible**, osfr will first open the item in your web browser so you can verify what is about to be deleted before proceeding.

If the project or component targeted for deletion contains sub-components, those must be deleted first. Setting `recurse = TRUE` will attempt to remove the hierarchy of sub-components before deleting the top-level entity.

Note: This functionality is limited to contributors with admin-level permissions.

Usage

```
osf_rm(x, recurse = FALSE, verbose = FALSE, check = TRUE)
```

Arguments

`x`	One of the following: • An `osf_tbl_node` with a single OSF project or component. • An `osf_tbl_file` containing a single directory or file.
`recurse`	Remove all sub-components before deleting the top-level entity. This only applies when deleting projects or components.
`verbose`	Logical, indicating whether to print informative messages about interactions with the OSF API (default `FALSE`).
`check`	If `FALSE` deletion will proceed without opening the item or requesting verification—this effectively removes your safety net.

Value

Invisibly returns `TRUE` if deletion was successful.

See Also

Other OSF file operations: `osf_cp()`, `osf_mkdir()`, `osf_mv()`

Examples

```
## Not run:
project <- osf_create_project("My Short-Lived Project")
osf_rm(project)

## End(Not run)
```

`osf_tbl` *OSF Tibbles*

Description

Items retrieved from OSF are represented as `osf_tbl` objects, specialized data frames based on the tibble class. See below for additional details.

Details

Each row of an `osf_tbl` represents a single OSF entity. This could be a user, project, component, directory, or file. An `osf_tbl` must include the following 3 columns:

1. `name`: indicates the name or title of the entity.
2. `id`: the unique identifier assigned by OSF.
3. `meta`: a list-column that stores the processed response returned by OSF's API. See the *Meta column* section below for more information.

Subclasses

`osf_tbl` is the parent class of 3 subclasses that are used to represent each of OSF's main entities:

1. `osf_tbl_user` for users.
2. `osf_tbl_file` for files and directories.
3. `osf_tbl_node` for projects and components.

OSF nodes

Projects and components are both implemented as *nodes* on OSF. The only distinction between the two is that a project is a top-level node, and a component must have a parent node (i.e., must be a sub-component of another project or component). Because projects and components are functionally identical, osfr uses the same `osf_tbl_node` class to represent both.

Meta column

The `meta` column contains all of the information returned from OSF's API for a single entity, structured as a named list with 3 elements:

1. `attributes` contains metadata about the entity (e.g., names, descriptions, tags, etc).
2. `links` contains urls to API endpoints with alternative representations of the entity or actions that may be performed on the entity.
3. `relationships` contains URLs to other entities with relationships to the entity (e.g., collaborators attached to a project).

This information is critical for `osfr`'s internal functions and should not be altered by users. For even more information about these elements, see OSF's API documentation.

Acknowledgments

Our implementation of the `osf_tbl` class is based on `dribble` objects from the googledrive package.

`osf_upload` *Upload files to OSF*

Description

Upload local files to a project, component, or directory on OSF.

Usage

```
osf_upload(
  x,
  path,
  recurse = FALSE,
  conflicts = "error",
  progress = FALSE,
  verbose = FALSE
)
```

Arguments

`x`	The upload destination on OSF. Can be one of the following: • An `osf_tbl_node` with a single project or component. • An `osf_tbl_file` with a single directory.
`path`	A character vector of paths pointing to existing local files and/directories.
`recurse`	If `TRUE`, fully recurse directories included in `path`. You can also control the number of levels to recurse by specifying a positive number.
`conflicts`	This determines what happens when a file with the same name exists at the specified destination. Can be one of the following:

- "error" (the default): throw an error and abort the file transfer operation.
- "skip": skip the conflicting file(s) and continue transferring the remaining files.
- "overwrite": replace the existing file with the transferred copy.

progress — Logical, if TRUE progress bars are displayed for each file transfer. Mainly useful for transferring large files. For tracking lots of small files, setting verbose = TRUE is more informative.

verbose — Logical, indicating whether to print informative messages about interactions with the OSF API (default FALSE).

Value

An osf_tbl_file containing the uploaded files and directories that were directly specified in path.

File and directory paths

The x argument indicates *where* on OSF the files will be uploaded (*i.e.*, the destination). The path argument indicates *what* will be uploaded, which can include a combination of files *and* directories.

When path points to a local file, the file is uploaded to the *root* of the specified OSF destination, regardless of where it's on your local machine (*i.e.*, the intermediate paths are not preserved). For example, the following would would upload both a.txt and b.txt to the root of my_proj:

```
osf_upload(my_proj, c("a.txt", "subdir/b.txt"))`
```

When path points to a local directory, a corresponding directory will be created at the root of the OSF destination, x, and any files within the local directory are uploaded to the new OSF directory. Therefore, we could maintain the directory structure in the above example by passing b.txt's parent directory to path instead of the file itself:

```
osf_upload(my_proj, c("a.txt", "subdir2"))
```

Likewise, osf_upload(my_proj, path = ".") will upload your entire current working directory to the specified OSF destination.

Uploading to subdirectories

In order to upload directly to an existing OSF directory you would first need to retrieve the directory as an osf_tbl_file. This can be accomplished by passing the directory's unique identifier to osf_retrieve_file(), or, if you don't have the ID handy, you can use osf_ls_files() to retrieve the directory by name.

```
# search for the 'rawdata' subdirectory within top-level 'data' directory
target_dir <- osf_ls_files(my_proj, path = "data", pattern = "rawdata")
# upload 'a.txt' to data/rawdata/ on OSF
osf_upload(target_dir, path = "a.txt")
```

A note about synchronization

While `osf_download()` and `osf_upload()` allow you to conveniently shuttle files back and forth between OSF and your local machine, it's important to note that **they are not file synchronization functions**. In contrast to something like `rsync`, `osf_download()`/`osf_upload()` do not take into account a file's contents or modification time. Whether you're uploading or downloading, if `conflicts = "overwrite"`, osfr will overwrite the existing file regardless of whether it is the more recent copy. You have been warned.

See Also

- `osf_download()` for downloading files and directories from OSF.
- `osf_ls_files()` for listing files and directories on OSF.

Examples

```
## Not run:
# Create an example file to upload to our example project
write.csv(iris, file = "iris.csv")
project <- osf_create_project("Flower Data")

# Upload the first version
osf_upload(project,"iris.csv")

# Modify the data file, upload version 2, and view it on OSF
write.csv(subset(iris, Species != "setosa"), file = "iris.csv")
project %>%
  osf_upload("iris.csv", conflicts = "overwrite") %>%
  osf_open()

## End(Not run)
```

Index