

# Package: s3fs (via r-universe)

August 29, 2024

**Type** Package

**Title** 'Amazon Web Service S3' File System

**Version** 0.1.7

**Description** Access 'Amazon Web Service Simple Storage Service' ('S3')  
<<https://aws.amazon.com/s3/>> as if it were a file system.  
Interface based on the R package 'fs'.

**License** MIT + file LICENSE

**URL** <https://github.com/DyfanJones/s3fs>

**BugReports** <https://github.com/DyfanJones/s3fs/issues>

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**Collate** 'zzz.R' 'utils.R' 's3filesystem\_class.R' 'file\_system.R'  
'file\_system\_async.R' 'reexport\_fs.R'

**Depends** R (>= 3.6.0)

**Imports** curl, R6, data.table, fs, future, future.apply, lgr,  
paws.storage (>= 0.2.0), utils

**Suggests** covr, testthat (>= 3.1.4)

**Config/testthat/edition** 3

**Repository** <https://dyfanjones.r-universe.dev>

**RemoteUrl** <https://github.com/DyfanJones/s3fs>

**RemoteRef** HEAD

**RemoteSha** 069a8a0f33116bbb6fec288008281518e7f01b6b

## Contents

s3fs-package	2
copy	3
copy_async	4

create . . . . .	5
delete . . . . .	6
delete_async . . . . .	7
download . . . . .	7
download_async . . . . .	8
exists . . . . .	9
file_type . . . . .	10
info . . . . .	10
path . . . . .	13
path_manipulate . . . . .	14
permission . . . . .	15
S3FileSystem . . . . .	16
s3_bucket_delete . . . . .	31
s3_dir_ls_url . . . . .	31
s3_dir_tree . . . . .	32
s3_file_move . . . . .	32
s3_file_move_async . . . . .	33
s3_file_system . . . . .	34
s3_file_temp . . . . .	35
s3_file_url . . . . .	36
s3_file_version_info . . . . .	36
s3_path_join . . . . .	37
s3_path_split . . . . .	37
stream . . . . .	38
stream_async . . . . .	39
tag . . . . .	40
touch . . . . .	40
upload . . . . .	41
upload_async . . . . .	42
<b>Index</b>	<b>43</b>

---

s3fs-package

s3fs: 'Amazon Web Service S3' File System

---

## Description

Access 'Amazon Web Service Simple Storage Service' ('S3') <https://aws.amazon.com/s3/> as if it were a file system. Interface based on the R package 'fs'.

## Author(s)

**Maintainer:** Dyfan Jones <[dyfan.r.jones@gmail.com](mailto:dyfan.r.jones@gmail.com)>

**See Also**

Useful links:

- <https://github.com/DyfanJones/s3fs>
- Report bugs at <https://github.com/DyfanJones/s3fs/issues>

---

copy

*Copy files and directories*

---

**Description**

s3\_file\_copy copies files

s3\_dir\_copy copies the directory recursively to the new location

**Usage**

```
s3_file_copy(  
  path,  
  new_path,  
  max_batch = fs_bytes("100MB"),  
  overwrite = FALSE,  
  ...  
)  
  
s3_dir_copy(  
  path,  
  new_path,  
  max_batch = fs_bytes("100MB"),  
  overwrite = FALSE,  
  ...  
)
```

**Arguments**

path	(character): path to a local directory of file or a uri.
new_path	(character): path to a local directory of file or a uri.
max_batch	( <a href="#">fs_bytes</a> ): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_put_object</a>

**Value**

character vector of s3 uri paths

**Examples**

```
## Not run:
# Require AWS S3 credentials

temp_file = "temp.txt"
file.create(temp_file)

s3_file_copy(
  temp_file,
  "s3://MyBucket/temp_file.txt"
)

## End(Not run)
```

copy\_async

*Copy files and directories***Description**

s3\_file\_copy copies files

s3\_dir\_copy copies the directory recursively to the new location

**Usage**

```
s3_file_copy_async(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)

s3_dir_copy_async(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)
```

**Arguments**

path	(character): path to a local directory of file or a uri.
new_path	(character): path to a local directory of file or a uri.
max_batch	( <a href="#">fs_bytes</a> ): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_put_object</a>

**Value**

return [future](#) object of [s3\\_file\\_copy\(\)](#), [s3\\_dir\\_copy\(\)](#)

**See Also**

[future](#) [s3\\_file\\_copy\(\)](#) [s3\\_dir\\_copy\(\)](#)

---

create	<i>Create files and directories</i>
--------	-------------------------------------

---

**Description**

[s3\\_file\\_create](#) create file on AWS S3, if file already exists it will be left unchanged.

[s3\\_dir\\_create](#) create empty directory of AWS S3.

**Usage**

```
s3_file_create(path, overwrite = FALSE, ...)
```

```
s3_bucket_create(
  path,
  region_name = NULL,
  mode = c("private", "public-read", "public-read-write", "authenticated-read"),
  versioning = FALSE,
  ...
)
```

```
s3_dir_create(path, overwrite = FALSE, ...)
```

**Arguments**

path	(character): A character vector of path or s3 uri.
overwrite	(logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_put_object</a> , <a href="#">s3_create_bucket</a>
region_name	(character): region for AWS S3 bucket, defaults to <a href="#">s3_file_system()</a> class region.
mode	(character): A character of the mode
versioning	(logical)

**Value**

character vector of s3 uri paths

## Examples

```
## Not run:
# Require AWS S3 credentials

temp_file = s3_file_temp(tmp_dir= "MyBucket")
s3_file_create(temp_file)

## End(Not run)
```

---

delete

*Delete files and directories*

---

## Description

s3\_file\_delete delete files in AWS S3

s3\_dir\_delete delete directories in AWS S3 recursively.

## Usage

```
s3_file_delete(path, ...)
```

```
s3_dir_delete(path)
```

## Arguments

path (character): A character vector of paths or s3 uris.

... parameters to be passed to [s3\\_delete\\_objects](#)

## Value

character vector of s3 uri paths

## Examples

```
## Not run:
# Require AWS S3 credentials

temp_file = s3_file_temp(tmp_dir= "MyBucket")
s3_file_create(temp_file)

s3_file_delete(temp_file)

## End(Not run)
```

---

delete_async	<i>Delete files and directories</i>
--------------	-------------------------------------

---

**Description**

s3\_file\_delete delete files in AWS S3  
s3\_dir\_delete delete directories in AWS S3 recursively.

**Usage**

s3\_file\_delete\_async(path, ...)  
s3\_dir\_delete\_async(path)

**Arguments**

path (character): A character vector of paths or s3 uris.  
... parameters to be passed to [s3\\_delete\\_objects](#)

**Value**

return [future](#) object of [s3\\_file\\_delete\(\)](#) [s3\\_dir\\_delete\(\)](#)

**See Also**

[future](#) [s3\\_file\\_delete\(\)](#) [s3\\_dir\\_delete\(\)](#)

---

download	<i>Download files and directories</i>
----------	---------------------------------------

---

**Description**

s3\_file\_download downloads AWS S3 files to local  
s3\_dir\_download downloads AWS s3 directory to local

**Usage**

s3\_file\_download(path, new\_path, overwrite = FALSE, ...)  
s3\_dir\_download(path, new\_path, overwrite = FALSE, ...)

**Arguments**

path (character): A character vector of paths or uris

new\_path (character): A character vector of paths to the new locations.

overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_get\\_object](#)

**Value**

character vector of s3 uri paths

**Examples**

```
## Not run:
# Require AWS S3 credentials

temp_file = s3_file_temp(tmp_dir= "MyBucket")
s3_file_create(temp_file)

s3_file_download(temp_file, "temp_file.txt")

## End(Not run)
```

---

download_async	<i>Download files and directories</i>
----------------	---------------------------------------

---

**Description**

s3\_file\_download downloads AWS S3 files to local

s3\_dir\_download downloads AWS S3 directory to local

**Usage**

```
s3_file_download_async(path, new_path, overwrite = FALSE, ...)

s3_dir_download_async(path, new_path, overwrite = FALSE, ...)
```

**Arguments**

path (character): A character vector of paths or uris

new\_path (character): A character vector of paths to the new locations.

overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_get\\_object](#)



**Value**

return `future` object of `s3_file_download()` `s3_dir_download()`

**See Also**

`future` `s3_file_download()` `s3_dir_download()`

---

`exists`*Download files and directories*

---

**Description**

`s3_file_exists` check if file exists in AWS S3

`s3_dir_exists` check if path is a directory in AWS S3

**Usage**

```
s3_file_exists(path)
```

```
s3_dir_exists(path = ".")
```

**Arguments**

`path` (character) s3 path to check

**Value**

logical vector if file exists

**Examples**

```
## Not run:  
# Require AWS S3 credentials  
  
temp_file = s3_file_temp(tmp_dir= "MyBucket")  
s3_file_create(temp_file)  
  
s3_file_exists(temp_file)  
  
## End(Not run)
```

---

file_type	<i>Functions to test for file types</i>
-----------	---

---

**Description**

Test for file types

**Usage**

```
s3_is_file(path)
s3_is_dir(path)
s3_is_bucket(path, ...)
s3_is_file_empty(path)
```

**Arguments**

path	(character): A character vector of paths or uris
...	parameters to be passed to <a href="#">s3_list_objects_v2</a>

---

info	<i>Get files and directories information</i>
------	--

---

**Description**

s3\_file\_info returns file information within AWS S3 directory  
s3\_file\_size returns file size in bytes  
s3\_dir\_info returns file name information within AWS S3 directory  
s3\_dir\_ls returns file name within AWS S3 directory

**Usage**

```
s3_file_info(path)

s3_file_size(path)

s3_dir_info(
  path = ".",
  type = c("any", "bucket", "directory", "file"),
  glob = NULL,
  regexp = NULL,
  invert = FALSE,
```

```

    recurse = FALSE,
    refresh = FALSE,
    ...
)

s3_dir_ls(
  path = ".",
  type = c("any", "bucket", "directory", "file"),
  glob = NULL,
  regexp = NULL,
  invert = FALSE,
  recurse = FALSE,
  refresh = FALSE,
  ...
)

```

### Arguments

path	(character): A character vector of one or more paths. Can be path or s3 uri.
type	(character): File type(s) to return. Default ("any") returns all AWS S3 object types.
glob	(character): A wildcard pattern (e.g. *.csv), passed onto <code>grep()</code> to filter paths.
regexp	(character): A regular expression (e.g. [.]csv\$), passed onto <code>grep()</code> to filter paths.
invert	(logical): If code return files which do not match.
recurse	(logical): Returns all AWS S3 objects in lower sub directories
refresh	(logical): Refresh cached in <code>s3_cache</code> .
...	parameters to be passed to <a href="#">s3_list_objects_v2</a>

### Value

`s3_file_info` A data.table with metadata for each file. Columns returned are as follows.

- `bucket_name` (character): AWS S3 bucket of file
- `key` (character): AWS S3 path key of file
- `uri` (character): S3 uri of file
- `size` (numeric): file size in bytes
- `type` (character): file type (file or directory)
- `etag` (character): An entity tag is an opaque identifier
- `last_modified` (POSIXct): Created date of file.
- `delete_marker` (logical): Specifies retrieved a logical marker
- `accept_ranges` (character): Indicates that a range of bytes was specified.
- `expiration` (character): File expiration
- `restore` (character): If file is archived

- `archive_status` (character): Archive status
- `missing_meta` (integer): Number of metadata entries not returned in "x-amz-meta" headers
- `version_id` (character): version id of file
- `cache_control` (character): caching behaviour for the request/reply chain
- `content_disposition` (character): presentational information of file
- `content_encoding` (character): file content encodings
- `content_language` (character): what language the content is in
- `content_type` (character): file MIME type
- `expires` (POSIXct): date and time the file is no longer cacheable
- `website_redirect_location` (character): redirects request for file to another
- `server_side_encryption` (character): File server side encryption
- `metadata` (list): metadata of file
- `sse_customer_algorithm` (character): server-side encryption with a customer-provided encryption key
- `sse_customer_key_md5` (character): server-side encryption with a customer-provided encryption key
- `ssekms_key_id` (character): ID of the Amazon Web Services Key Management Service
- `bucket_key_enabled` (logical): s3 bucket key for server-side encryption with
- `storage_class` (character): file storage class information
- `request_charged` (character): indicates successfully charged for request
- `replication_status` (character): return specific header if request involves a bucket that is either a source or a destination in a replication rule [https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#S3.Client.head\\_object](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#S3.Client.head_object)
- `parts_count` (integer): number of count parts the file has
- `object_lock_mode` (character): the file lock mode
- `object_lock_retain_until_date` (POSIXct): date and time of when `object_lock_mode` expires
- `object_lock_legal_hold_status` (character): file legal holding

`s3_dir_info` data.table with directory metadata

- `bucket_name` (character): AWS S3 bucket of file
- `key` (character): AWS S3 path key of file
- `uri` (character): S3 uri of file
- `size` (numeric): file size in bytes
- `version_id` (character): version id of file
- `etag` (character): An entity tag is an opaque identifier
- `last_modified` (POSIXct): Created date of file

`s3_dir_ls` character vector of s3 uri paths

**Examples**

```
## Not run:
# Require AWS S3 credentials

temp_file = s3_file_temp(tmp_dir= "MyBucket")
s3_file_create(temp_file)

s3_file_info(temp_file)

## End(Not run)
```

---

path	<i>Construct path for file or directory</i>
------	---

---

**Description**

Constructs a s3 uri path

**Usage**

```
s3_path(..., ext = "")
```

**Arguments**

...	(character): Character vectors
ext	(character): An optional extension to append to the generated path

**Value**

character vector of s3 uri paths

**Examples**

```
## Not run:
# Require AWS S3 credentials

s3_path("my_bucket1", "my_bucket2")

## End(Not run)
```

---

path_manipulate	<i>Manipulate s3 uri paths</i>
-----------------	--------------------------------

---

## Description

s3\_path\_dir returns the directory portion of s3 uri

s3\_path\_file returns the file name portion of the s3 uri path

s3\_path\_ext returns the last extension for a path.

s3\_path\_ext\_remove removes the last extension and return the rest of the s3 uri.

s3\_path\_ext\_set replace the extension with a new extension.

## Usage

s3\_path\_dir(path)

s3\_path\_file(path)

s3\_path\_ext(path)

s3\_path\_ext\_remove(path)

s3\_path\_ext\_set(path, ext)

## Arguments

path (character): A character vector of paths

ext (character): New file extension

## Examples

```
## Not run:  
# Require AWS S3 credentials  
  
s3_path_dir("s3://my_bucket1/hi.txt")  
  
s3_path_file("s3://my_bucket1/hi.txt")  
  
## End(Not run)
```

---

permission	<i>Change file permissions</i>
------------	--------------------------------

---

**Description**

Change file permissions

**Usage**

```
s3_file_chmod(  
  path,  
  mode = c("private", "public-read", "public-read-write", "authenticated-read",  
           "aws-exec-read", "bucket-owner-read", "bucket-owner-full-control")  
)  
  
s3_bucket_chmod(  
  path,  
  mode = c("private", "public-read", "public-read-write", "authenticated-read")  
)
```

**Arguments**

path	(character): A character vector of path or s3 uri.
mode	(character): A character of the mode

**Value**

character vector of s3 uri paths

**Examples**

```
## Not run:  
# Require AWS S3 credentials  
  
temp_file = s3_file_temp(tmp_dir = "MyBucket")  
s3_file_create(temp_file)  
  
# Reset connection to connect to a different region  
s3_file_chmod(  
  profile_name = "s3fs_example",  
  region_name = "us-east-1",  
  refresh = TRUE  
)  
  
## End(Not run)
```

---

S3FileSystem

Access AWS S3 as if it were a file system.

---

### Description

This creates a file system "like" API based off fs (e.g. dir\_ls, file\_copy, etc.) for AWS S3 storage.

### Public fields

s3\_cache Cache AWS S3  
s3\_cache\_bucket Cached s3 bucket  
s3\_client paws s3 client  
region\_name AWS region when creating new connections  
profile\_name The name of a profile to use  
multipart\_threshold Threshold to use multipart  
request\_payer Threshold to use multipart  
pid Get the process ID of the R Session

### Active bindings

retries number of retries

### Methods

#### Public methods:

- `S3FileSystem$new()`
- `S3FileSystem$file_chmod()`
- `S3FileSystem$file_copy()`
- `S3FileSystem$file_create()`
- `S3FileSystem$file_delete()`
- `S3FileSystem$file_download()`
- `S3FileSystem$file_exists()`
- `S3FileSystem$file_info()`
- `S3FileSystem$file_move()`
- `S3FileSystem$file_size()`
- `S3FileSystem$file_stream_in()`
- `S3FileSystem$file_stream_out()`
- `S3FileSystem$file_temp()`
- `S3FileSystem$file_tag_delete()`
- `S3FileSystem$file_tag_info()`
- `S3FileSystem$file_tag_update()`
- `S3FileSystem$file_touch()`



- S3FileSystem\$file\_upload()
- S3FileSystem\$file\_url()
- S3FileSystem\$file\_version\_info()
- S3FileSystem\$is\_file()
- S3FileSystem\$is\_dir()
- S3FileSystem\$is\_bucket()
- S3FileSystem\$is\_file\_empty()
- S3FileSystem\$bucket\_chmod()
- S3FileSystem\$bucket\_create()
- S3FileSystem\$bucket\_delete()
- S3FileSystem\$dir\_copy()
- S3FileSystem\$dir\_create()
- S3FileSystem\$dir\_delete()
- S3FileSystem\$dir\_exists()
- S3FileSystem\$dir\_download()
- S3FileSystem\$dir\_info()
- S3FileSystem\$dir\_ls()
- S3FileSystem\$dir\_ls\_url()
- S3FileSystem\$dir\_tree()
- S3FileSystem\$dir\_upload()
- S3FileSystem\$path()
- S3FileSystem\$path\_dir()
- S3FileSystem\$path\_ext()
- S3FileSystem\$path\_ext\_remove()
- S3FileSystem\$path\_ext\_set()
- S3FileSystem\$path\_file()
- S3FileSystem\$path\_join()
- S3FileSystem\$path\_split()
- S3FileSystem\$clear\_cache()
- S3FileSystem\$clone()

**Method new():** Initialize S3FileSystem class

*Usage:*

```
S3FileSystem$new(
  aws_access_key_id = NULL,
  aws_secret_access_key = NULL,
  aws_session_token = NULL,
  region_name = NULL,
  profile_name = NULL,
  endpoint = NULL,
  disable_ssl = FALSE,
  multipart_threshold = fs_bytes("2GB"),
  request_payer = FALSE,
  anonymous = FALSE,
  ...
)
```

*Arguments:*

aws\_access\_key\_id (character): AWS access key ID  
 aws\_secret\_access\_key (character): AWS secret access key  
 aws\_session\_token (character): AWS temporary session token  
 region\_name (character): Default region when creating new connections  
 profile\_name (character): The name of a profile to use. If not given, then the default profile is used.  
 endpoint (character): The complete URL to use for the constructed client.  
 disable\_ssl (logical): Whether or not to use SSL. By default, SSL is used.  
 multipart\_threshold ([fs\\_bytes](#)): Threshold to use multipart instead of standard copy and upload methods.  
 request\_payer (logical): Confirms that the requester knows that they will be charged for the request.  
 anonymous (logical): Set up anonymous credentials when connecting to AWS S3.  
 ... Other parameters within paws client.

**Method** file\_chmod(): Change file permissions*Usage:*

```
S3FileSystem$file_chmod(
  path,
  mode = c("private", "public-read", "public-read-write", "authenticated-read",
    "aws-exec-read", "bucket-owner-read", "bucket-owner-full-control")
)
```

*Arguments:*

path (character): A character vector of path or s3 uri.  
 mode (character): A character of the mode

*Returns:* character vector of s3 uri paths

**Method** file\_copy(): copy files*Usage:*

```
S3FileSystem$file_copy(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)
```

*Arguments:*

path (character): path to a local directory of file or a uri.  
 new\_path (character): path to a local directory of file or a uri.  
 max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.  
 overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.  
 ... parameters to be passed to [s3\\_put\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `file_create()`: Create file on AWS S3, if file already exists it will be left unchanged.

*Usage:*

```
S3FileSystem$file_create(path, overwrite = FALSE, ...)
```

*Arguments:*

`path` (character): A character vector of path or s3 uri.

`overwrite` (logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.

`...` parameters to be passed to [s3\\_put\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `file_delete()`: Delete files in AWS S3

*Usage:*

```
S3FileSystem$file_delete(path, ...)
```

*Arguments:*

`path` (character): A character vector of paths or s3 uris.

`...` parameters to be passed to [s3\\_delete\\_objects](#)

*Returns:* character vector of s3 uri paths

**Method** `file_download()`: Downloads AWS S3 files to local

*Usage:*

```
S3FileSystem$file_download(path, new_path, overwrite = FALSE, ...)
```

*Arguments:*

`path` (character): A character vector of paths or uris

`new_path` (character): A character vector of paths to the new locations.

`overwrite` (logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.

`...` parameters to be passed to [s3\\_get\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `file_exists()`: Check if file exists in AWS S3

*Usage:*

```
S3FileSystem$file_exists(path)
```

*Arguments:*

`path` (character) s3 path to check

*Returns:* logical vector if file exists

**Method** `file_info()`: Returns file information within AWS S3 directory

*Usage:*

```
S3FileSystem$file_info(path)
```

*Arguments:*

`path` (character): A character vector of paths or uris.

*Returns:* A data.table with metadata for each file. Columns returned are as follows.

- `bucket_name` (character): AWS S3 bucket of file
- `key` (character): AWS S3 path key of file
- `uri` (character): S3 uri of file
- `size` (numeric): file size in bytes
- `type` (character): file type (file or directory)
- `etag` (character): An entity tag is an opaque identifier
- `last_modified` (POSIXct): Created date of file.
- `delete_marker` (logical): Specifies retrieved a logical marker
- `accept_ranges` (character): Indicates that a range of bytes was specified.
- `expiration` (character): File expiration
- `restore` (character): If file is archived
- `archive_status` (character): Archive status
- `missing_meta` (integer): Number of metadata entries not returned in "x-amz-meta" headers
- `version_id` (character): version id of file
- `cache_control` (character): caching behaviour for the request/reply chain
- `content_disposition` (character): presentational information of file
- `content_encoding` (character): file content encodings
- `content_language` (character): what language the content is in
- `content_type` (character): file MIME type
- `expires` (POSIXct): date and time the file is no longer cacheable
- `website_redirect_location` (character): redirects request for file to another
- `server_side_encryption` (character): File server side encryption
- `metadata` (list): metadata of file
- `sse_customer_algorithm` (character): server-side encryption with a customer-provided encryption key
- `sse_customer_key_md5` (character): server-side encryption with a customer-provided encryption key
- `ssekms_key_id` (character): ID of the Amazon Web Services Key Management Service
- `bucket_key_enabled` (logical): s3 bucket key for server-side encryption with
- `storage_class` (character): file storage class information
- `request_charged` (character): indicates successfully charged for request
- `replication_status` (character): return specific header if request involves a bucket that is either a source or a destination in a replication rule [https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#S3.Client.head\\_object](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#S3.Client.head_object)
- `parts_count` (integer): number of count parts the file has
- `object_lock_mode` (character): the file lock mode
- `object_lock_retain_until_date` (POSIXct): date and time of when `object_lock_mode` expires
- `object_lock_legal_hold_status` (character): file legal holding

**Method** `file_move()`: Move files to another location on AWS S3

*Usage:*

```

S3FileSystem$file_move(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)

```

*Arguments:*

path (character): A character vector of s3 uri

new\_path (character): A character vector of s3 uri.

max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.

overwrite (logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_copy\\_object](#)

*Returns:* character vector of s3 uri paths

**Method file\_size():** Return file size in bytes

*Usage:*

```
S3FileSystem$file_size(path)
```

*Arguments:*

path (character): A character vector of s3 uri

**Method file\_stream\_in():** Streams in AWS S3 file as a raw vector

*Usage:*

```
S3FileSystem$file_stream_in(path, ...)
```

*Arguments:*

path (character): A character vector of paths or s3 uri

... parameters to be passed to [s3\\_get\\_object](#)

*Returns:* list of raw vectors containing the contents of the file

**Method file\_stream\_out():** Streams out raw vector to AWS S3 file

*Usage:*

```

S3FileSystem$file_stream_out(
  obj,
  path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)

```

*Arguments:*

obj (raw|character): A raw vector, rawConnection, url to be streamed up to AWS S3.

path (character): A character vector of paths or s3 uri

max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.

overwrite (logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_put\\_object](#)

Returns: character vector of s3 uri paths

**Method file\_temp():** return the name which can be used as a temporary file

Usage:

```
S3FileSystem$file_temp(pattern = "file", tmp_dir = "", ext = "")
```

Arguments:

pattern (character): A character vector with the non-random portion of the name.

tmp\_dir (character): The directory the file will be created in.

ext (character): A character vector of one or more paths.

Returns: character vector of s3 uri paths

**Method file\_tag\_delete():** Delete file tags

Usage:

```
S3FileSystem$file_tag_delete(path)
```

Arguments:

path (character): A character vector of paths or s3 uri

... parameters to be passed to [s3\\_put\\_object](#)

Returns: character vector of s3 uri paths

**Method file\_tag\_info():** Get file tags

Usage:

```
S3FileSystem$file_tag_info(path)
```

Arguments:

path (character): A character vector of paths or s3 uri

Returns: data.table of file version metadata

- bucket\_name (character): AWS S3 bucket of file
- key (character): AWS S3 path key of file
- uri (character): S3 uri of file
- size (numeric): file size in bytes
- version\_id (character): version id of file
- tag\_key (character): name of tag
- tag\_value (character): tag value

**Method file\_tag\_update():** Update file tags

Usage:

```
S3FileSystem$file_tag_update(path, tags, overwrite = FALSE)
```

Arguments:

path (character): A character vector of paths or s3 uri

tags (list): Tags to be applied

overwrite (logical): To overwrite tagging or to modify inplace. Default will modify inplace.

*Returns:* character vector of s3 uri paths

**Method** `file_touch()`: Similar to `fs::file_touch` this does not create the file if it does not exist. Use `s3fs$file_create()` to do this if needed.

*Usage:*

```
S3FileSystem$file_touch(path, ...)
```

*Arguments:*

path (character): A character vector of paths or s3 uri

... parameters to be passed to [s3\\_copy\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `file_upload()`: Uploads files to AWS S3

*Usage:*

```
S3FileSystem$file_upload(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)
```

*Arguments:*

path (character): A character vector of local file paths to upload to AWS S3

new\_path (character): A character vector of AWS S3 paths or uri's of the new locations.

max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.

overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_put\\_object](#) and [s3\\_create\\_multipart\\_upload](#)

*Returns:* character vector of s3 uri paths

**Method** `file_url()`: Generate presigned url for S3 object

*Usage:*

```
S3FileSystem$file_url(path, expiration = 3600L, ...)
```

*Arguments:*

path (character): A character vector of paths or uris

expiration (numeric): The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)

... parameters passed to [s3\\_get\\_object](#)

*Returns:* return character of urls

**Method** `file_version_info()`: Get file versions

*Usage:*

```
S3FileSystem$file_version_info(path, ...)
```

*Arguments:*

path (character): A character vector of paths or uris

... parameters to be passed to [s3\\_list\\_object\\_versions](#)

*Returns:* return data.table with file version info, columns below:

- bucket\_name (character): AWS S3 bucket of file
- key (character): AWS S3 path key of file
- uri (character): S3 uri of file
- size (numeric): file size in bytes
- version\_id (character): version id of file
- owner (character): file owner
- etag (character): An entity tag is an opaque identifier
- last\_modified (POSIXct): Created date of file.

**Method is\_file():** Test for file types*Usage:*

```
S3FileSystem$is_file(path)
```

*Arguments:*

path (character): A character vector of paths or uris

*Returns:* logical vector if object is a file

**Method is\_dir():** Test for file types*Usage:*

```
S3FileSystem$is_dir(path)
```

*Arguments:*

path (character): A character vector of paths or uris

*Returns:* logical vector if object is a directory

**Method is\_bucket():** Test for file types*Usage:*

```
S3FileSystem$is_bucket(path, ...)
```

*Arguments:*

path (character): A character vector of paths or uris

... parameters to be passed to [s3\\_list\\_objects\\_v2](#)

*Returns:* logical vector if object is a AWS S3 bucket

**Method is\_file\_empty():** Test for file types*Usage:*

```
S3FileSystem$is_file_empty(path)
```

*Arguments:*

path (character): A character vector of paths or uris

*Returns:* logical vector if file is empty



**Method** bucket\_chmod(): Change bucket permissions

*Usage:*

```
S3FileSystem$bucket_chmod(  
  path,  
  mode = c("private", "public-read", "public-read-write", "authenticated-read")  
)
```

*Arguments:*

path (character): A character vector of path or s3 uri.

mode (character): A character of the mode

*Returns:* character vector of s3 uri paths

**Method** bucket\_create(): Create bucket

*Usage:*

```
S3FileSystem$bucket_create(  
  path,  
  region_name = NULL,  
  mode = c("private", "public-read", "public-read-write", "authenticated-read"),  
  versioning = FALSE,  
  ...  
)
```

*Arguments:*

path (character): A character vector of path or s3 uri.

region\_name (character): aws region

mode (character): A character of the mode

versioning (logical): Whether to set the bucket to versioning or not.

... parameters to be passed to [s3\\_create\\_bucket](#)

*Returns:* character vector of s3 uri paths

**Method** bucket\_delete(): Delete bucket

*Usage:*

```
S3FileSystem$bucket_delete(path)
```

*Arguments:*

path (character): A character vector of path or s3 uri.

**Method** dir\_copy(): Copies the directory recursively to the new location.

*Usage:*

```
S3FileSystem$dir_copy(  
  path,  
  new_path,  
  max_batch = fs_bytes("100MB"),  
  overwrite = FALSE,  
  ...  
)
```

*Arguments:*

`path` (character): path to a local directory of file or a uri.

`new_path` (character): path to a local directory of file or a uri.

`max_batch` ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.

`overwrite` (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_put\\_object](#) and [s3\\_create\\_multipart\\_upload](#)

*Returns:* character vector of s3 uri paths

**Method** `dir_create()`: Create empty directory*Usage:*

```
S3FileSystem$dir_create(path, overwrite = FALSE, ...)
```

*Arguments:*

`path` (character): A vector of directory or uri to be created in AWS S3

`overwrite` (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_put\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `dir_delete()`: Delete contents and directory in AWS S3*Usage:*

```
S3FileSystem$dir_delete(path)
```

*Arguments:*

`path` (character): A vector of paths or uris to directories to be deleted.

*Returns:* character vector of s3 uri paths

**Method** `dir_exists()`: Check if path exists in AWS S3*Usage:*

```
S3FileSystem$dir_exists(path = ".")
```

*Arguments:*

`path` (character) aws s3 path to be checked

*Returns:* character vector of s3 uri paths

**Method** `dir_download()`: Downloads AWS S3 files to local*Usage:*

```
S3FileSystem$dir_download(path, new_path, overwrite = FALSE, ...)
```

*Arguments:*

`path` (character): A character vector of paths or uris

`new_path` (character): A character vector of paths to the new locations. Please ensure directories end with a /.

`overwrite` (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

... parameters to be passed to [s3\\_get\\_object](#)

*Returns:* character vector of s3 uri paths

**Method** `dir_info()`: Returns file information within AWS S3 directory

*Usage:*

```
S3FileSystem$dir_info(
  path = ".",
  type = c("any", "bucket", "directory", "file"),
  glob = NULL,
  regexp = NULL,
  invert = FALSE,
  recurse = FALSE,
  refresh = FALSE,
  ...
)
```

*Arguments:*

`path` (character): A character vector of one or more paths. Can be path or s3 uri.

`type` (character): File type(s) to return. Default ("any") returns all AWS S3 object types.

`glob` (character): A wildcard pattern (e.g. \*.csv), passed onto `grep()` to filter paths.

`regexp` (character): A regular expression (e.g. [.]csv\$), passed onto `grep()` to filter paths.

`invert` (logical): If code return files which do not match.

`recurse` (logical): Returns all AWS S3 objects in lower sub directories

`refresh` (logical): Refresh cached in s3\_cache.

... parameters to be passed to [s3\\_list\\_objects\\_v2](#)

*Returns:* data.table with directory metadata

- `bucket_name` (character): AWS S3 bucket of file
- `key` (character): AWS S3 path key of file
- `uri` (character): S3 uri of file
- `size` (numeric): file size in bytes
- `version_id` (character): version id of file
- `etag` (character): An entity tag is an opaque identifier
- `last_modified` (POSIXct): Created date of file

**Method** `dir_ls()`: Returns file name within AWS S3 directory

*Usage:*

```
S3FileSystem$dir_ls(
  path = ".",
  type = c("any", "bucket", "directory", "file"),
  glob = NULL,
  regexp = NULL,
  invert = FALSE,
  recurse = FALSE,
  refresh = FALSE,
  ...
)
```

*Arguments:*

path (character): A character vector of one or more paths. Can be path or s3 uri.  
 type (character): File type(s) to return. Default ("any") returns all AWS S3 object types.  
 glob (character): A wildcard pattern (e.g. \*.csv), passed onto `grep()` to filter paths.  
 regexp (character): A regular expression (e.g. [. ]csv\$), passed onto `grep()` to filter paths.  
 invert (logical): If code return files which do not match.  
 recurse (logical): Returns all AWS S3 objects in lower sub directories  
 refresh (logical): Refresh cached in s3\_cache.  
 ... parameters to be passed to [s3\\_list\\_objects\\_v2](#)

*Returns:* character vector of s3 uri paths

**Method** `dir_ls_url()`: Generate presigned url to list S3 directories

*Usage:*

```
S3FileSystem$dir_ls_url(path, expiration = 3600L, recurse = FALSE, ...)
```

*Arguments:*

path (character): A character vector of paths or uris  
 expiration (numeric): The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)  
 recurse (logical): Returns all AWS S3 objects in lower sub directories  
 ... parameters passed to [s3\\_list\\_objects\\_v2](#)

*Returns:* return character of urls

**Method** `dir_tree()`: Print contents of directories in a tree-like format

*Usage:*

```
S3FileSystem$dir_tree(path, recurse = TRUE, ...)
```

*Arguments:*

path (character): path A path to print the tree from  
 recurse (logical): Returns all AWS S3 objects in lower sub directories  
 ... Additional arguments passed to [s3\\_dir\\_ls](#).

*Returns:* character vector of s3 uri paths

**Method** `dir_upload()`: Uploads local directory to AWS S3

*Usage:*

```
S3FileSystem$dir_upload(  
  path,  
  new_path,  
  max_batch = fs_bytes("100MB"),  
  overwrite = FALSE,  
  ...  
)
```

*Arguments:*

path (character): A character vector of local file paths to upload to AWS S3

`new_path` (character): A character vector of AWS S3 paths or uri's of the new locations.  
`max_batch` ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.  
`overwrite` (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.  
... parameters to be passed to [s3\\_put\\_object](#) and [s3\\_create\\_multipart\\_upload](#)  
*Returns:* character vector of s3 uri paths

**Method** `path()`: Constructs a s3 uri path

*Usage:*

```
S3FileSystem$path(..., ext = "")
```

*Arguments:*

... (character): Character vectors

`ext` (character): An optional extension to append to the generated path

*Returns:* character vector of s3 uri paths

**Method** `path_dir()`: Returns the directory portion of s3 uri

*Usage:*

```
S3FileSystem$path_dir(path)
```

*Arguments:*

`path` (character): A character vector of paths

*Returns:* character vector of s3 uri paths

**Method** `path_ext()`: Returns the last extension for a path.

*Usage:*

```
S3FileSystem$path_ext(path)
```

*Arguments:*

`path` (character): A character vector of paths

*Returns:* character s3 uri file extension

**Method** `path_ext_remove()`: Removes the last extension and return the rest of the s3 uri.

*Usage:*

```
S3FileSystem$path_ext_remove(path)
```

*Arguments:*

`path` (character): A character vector of paths

*Returns:* character vector of s3 uri paths

**Method** `path_ext_set()`: Replace the extension with a new extension.

*Usage:*

```
S3FileSystem$path_ext_set(path, ext)
```

*Arguments:*

`path` (character): A character vector of paths

`ext` (character): New file extension

*Returns:* character vector of s3 uri paths

**Method** `path_file()`: Returns the file name portion of the s3 uri path

*Usage:*

`S3FileSystem$path_file(path)`

*Arguments:*

`path` (character): A character vector of paths

*Returns:* character vector of file names

**Method** `path_join()`: Construct an s3 uri path from path vector

*Usage:*

`S3FileSystem$path_join(parts)`

*Arguments:*

`parts` (character): A character vector of one or more paths

*Returns:* character vector of s3 uri paths

**Method** `path_split()`: Split s3 uri path to core components bucket, key and version id

*Usage:*

`S3FileSystem$path_split(path)`

*Arguments:*

`path` (character): A character vector of one or more paths or s3 uri

*Returns:* list character vectors splitting the s3 uri path in "Bucket", "Key" and "VersionId"

**Method** `clear_cache()`: Clear S3 Cache

*Usage:*

`S3FileSystem$clear_cache(path = NULL)`

*Arguments:*

`path` (character): s3 path to be cl

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`S3FileSystem$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

## Note

This method will only update the modification time of the AWS S3 object.

---

s3_bucket_delete	Delete bucket
------------------	---------------

---

**Description**

Delete AWS S3 bucket including all objects in the bucket itself.

**Usage**

s3\_bucket\_delete(path)

**Arguments**

path (character): A character vector of path or s3 uri.

---

s3_dir_ls_url	Generate presigned url to list S3 directories
---------------	---

---

**Description**

Generate presigned url to list S3 directories

**Usage**

s3\_dir\_ls\_url(path, expiration = 3600L, recurse = FALSE, ...)

**Arguments**

path (character): A character vector of paths or uris

expiration (numeric): The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)

recurse (logical): Returns all AWS S3 objects in lower sub directories

... parameters passed to [s3\\_list\\_objects\\_v2](#)

**Value**

return character of urls

---

s3_dir_tree	<i>Print contents of directories in a tree-like format</i>
-------------	--

---

**Description**

Print contents of directories in a tree-like format

**Usage**

```
s3_dir_tree(path, recurse = TRUE, ...)
```

**Arguments**

path	(character): path A path to print the tree from
recurse	(logical): Returns all AWS S3 objects in lower sub directories
...	Additional arguments passed to <a href="#">s3_dir_ls</a> .

**Value**

character vector of s3 uri paths

---

s3_file_move	<i>Move or rename S3 files</i>
--------------	--------------------------------

---

**Description**

Move files to another location on AWS S3

**Usage**

```
s3_file_move(path, new_path, max_batch = 100 * MB, overwrite = FALSE, ...)
```

**Arguments**

path	(character): A character vector of s3 uri
new_path	(character): A character vector of s3 uri.
max_batch	(numeric): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_copy_object</a>

**Value**

character vector of s3 uri paths



### Examples

```
## Not run:
# Require AWS S3 credentials

temp_file = s3_file_temp(tmp_dir= "MyBucket")
s3_file_create(temp_file)

s3_file_move(temp_file, "s3://MyBucket/new_file.txt")

## End(Not run)
```

---

s3_file_move_async	<i>Move or rename S3 files</i>
--------------------	--------------------------------

---

### Description

Move files to another location on AWS S3

### Usage

```
s3_file_move_async(
  path,
  new_path,
  max_batch = 100 * MB,
  overwrite = FALSE,
  ...
)
```

### Arguments

path	(character): A character vector of s3 uri
new_path	(character): A character vector of s3 uri.
max_batch	(numeric): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if they exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_copy_object</a>

### Value

return [future](#) object of [s3\\_file\\_move\(\)](#)

### See Also

[future](#) [s3\\_file\\_move\(\)](#)

s3\_file\_system

*Access AWS S3 as if it were a file system.***Description**

This creates a file system "like" API based off fs (e.g. `dir_ls`, `file_copy`, etc.) for AWS S3 storage. To set up AWS credentials please look at <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html>

**Usage**

```
s3_file_system(
  aws_access_key_id = NULL,
  aws_secret_access_key = NULL,
  aws_session_token = NULL,
  region_name = NULL,
  profile_name = NULL,
  endpoint = NULL,
  disable_ssl = FALSE,
  multipart_threshold = fs_bytes("2GB"),
  request_payer = FALSE,
  anonymous = FALSE,
  retries = 5,
  refresh = FALSE,
  ...
)
```

**Arguments**

<code>aws_access_key_id</code>	(character): AWS access key ID
<code>aws_secret_access_key</code>	(character): AWS secret access key
<code>aws_session_token</code>	(character): AWS temporary session token
<code>region_name</code>	(character): Default region when creating new connections
<code>profile_name</code>	(character): The name of a profile to use. If not given, then the default profile is used.
<code>endpoint</code>	(character): The complete URL to use for the constructed client.
<code>disable_ssl</code>	(logical): Whether or not to use SSL. By default, SSL is used.
<code>multipart_threshold</code>	( <a href="#">fs_bytes</a> ): Threshold to use multipart instead of standard copy and upload methods.
<code>request_payer</code>	(logical): Confirms that the requester knows that they will be charged for the request.

anonymous	(logical): Set up anonymous credentials when connecting to AWS S3.
retries	(numeric): max number of retry attempts
refresh	(logical): Refresh cached S3FileSystem class
...	Other parameters within paws client.

**Value**

S3FileSystem class invisible

**Examples**

```
## Not run:
# Require AWS S3 credentials

# Set up connection using profile
s3_file_system(profile_name = "s3fs_example")

# Reset connection to connect to a different region
s3_file_system(
  profile_name = "s3fs_example",
  region_name = "us-east-1",
  refresh = TRUE
)

## End(Not run)
```

---

s3_file_temp	<i>Create name for temporary files</i>
--------------	--

---

**Description**

return the name which can be used as a temporary file

**Usage**

```
s3_file_temp(pattern = "file", tmp_dir = "", ext = "")
```

**Arguments**

pattern	(character): A character vector with the non-random portion of the name.
tmp_dir	(character): The directory the file will be created in. By default the cached s3 bucket will be applied otherwise "" will be used.
ext	(character): A character vector of one or more paths.

**Value**

character vector of s3 uri paths

Examples

```
## Not run:
# Require AWS S3 credentials

s3_file_temp(tmp_dir = "MyBucket")

## End(Not run)
```

---

s3_file_url	<i>Generate presigned url for S3 object</i>
-------------	---

---

Description

Generate presigned url for S3 object

Usage

```
s3_file_url(path, expiration = 3600L, ...)
```

Arguments

- path (character): A character vector of paths or uris
- expiration (numeric): The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)
- ... parameters to be passed to params parameter of [s3\\_generate\\_presigned\\_url](#)

Value

return character of urls

---

s3_file_version_info	<i>Query file version metadata</i>
----------------------	------------------------------------

---

Description

Get file versions

Usage

```
s3_file_version_info(path, ...)
```

Arguments

- path (character): A character vector of paths or uris
- ... parameters to be passed to [s3\\_list\\_object\\_versions](#)

---

s3_path_join	<i>Construct AWS S3 path</i>
--------------	------------------------------

---

**Description**

Construct an s3 uri path from path vector

**Usage**

```
s3_path_join(path)
```

**Arguments**

path (character): A character vector of one or more paths

**Value**

character vector of s3 uri paths

**Examples**

```
## Not run:  
# Require AWS S3 credentials  
  
s3_path_dir(c("s3://my_bucket1/hi.txt", "s3://my_bucket/bye.txt"))  
  
## End(Not run)
```

---

s3_path_split	<i>Split s3 path and uri</i>
---------------	------------------------------

---

**Description**

Split s3 uri path to core components bucket, key and version id

**Usage**

```
s3_path_split(path)
```

**Arguments**

path (character): A character vector of one or more paths or s3 uri

**Value**

list character vectors splitting the s3 uri path in "Bucket", "Key" and "VersionId"

Examples

```
## Not run:
# Require AWS S3 credentials

s3_path_dir("s3://my_bucket1/hi.txt")

## End(Not run)
```

---

stream	<i>Streams data from R to AWS S3.</i>
--------	---------------------------------------

---

Description

s3\_file\_stream\_in streams in AWS S3 file as a raw vector  
s3\_file\_stream\_out streams raw vector out to AWS S3 file

Usage

```
s3_file_stream_in(path, ...)

s3_file_stream_out(
  obj,
  path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)
```

Arguments

- path (character): A character vector of paths or s3 uri
- ... parameters to be passed to [s3\\_get\\_object](#) and [s3\\_put\\_object](#)
- obj (raw|character): A raw vector, rawConnection, url to be streamed up to AWS S3.
- max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.
- overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

Value

list of raw vectors containing the contents of the file

**Examples**

```
## Not run:
# Require AWS S3 credentials

obj = list(charToRaw("contents1"), charToRaw("contents2"))

dir = s3_file_temp(tmp_dir = "MyBucket")
path = s3_path(dir, letters[1:2], ext = "txt")

s3_file_stream_out(obj, path)
s3_file_stream_in(path)

## End(Not run)
```

stream\_async

*Streams data from R to AWS S3.***Description**

s3\_file\_stream\_in streams in AWS S3 file as a raw vector  
 s3\_file\_stream\_out streams raw vector out to AWS S3 file

**Usage**

```
s3_file_stream_in_async(path, ...)

s3_file_stream_out_async(
  obj,
  path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)
```

**Arguments**

path	(character): A character vector of paths or s3 uri
...	parameters to be passed to <a href="#">s3_get_object</a> and <a href="#">s3_put_object</a>
obj	(raw character): A raw vector, rawConnection, url to be streamed up to AWS S3.
max_batch	( <a href="#">fs_bytes</a> ): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.

**Value**

return [future](#) object of [s3\\_file\\_stream\\_in\(\)](#) [s3\\_file\\_stream\\_out\(\)](#)

See Also

[future](#) [s3\\_file\\_move\(\)](#) [s3\\_file\\_stream\\_in\(\)](#) [s3\\_file\\_stream\\_out\(\)](#)

---

tag	<i>Modifying file tags</i>
-----	----------------------------

---

Description

`s3_file_tag_delete` delete file tags  
`s3_file_tag_info` get file tags  
`s3_file_tag_info`

Usage

```
s3_file_tag_delete(path)

s3_file_tag_info(path)

s3_file_tag_update(path, tags, overwrite = FALSE)
```

Arguments

path	(character): A character vector of paths or s3 uri
tags	(list): Tags to be applied
overwrite	(logical): To overwrite tagging or to modify inplace. Default will modify in-place.

---

touch	<i>Change file modification time</i>
-------	--------------------------------------

---

Description

Similar to `fs::file_touch` this does not create the file if it does not exist. Use [s3\\_file\\_create](#) to do this if needed.

Usage

```
s3_file_touch(path, ...)
```

Arguments

path	(character): A character vector of paths or s3 uri
...	parameters to be passed to <a href="#">s3_copy_object</a>



**Value**

character vector of s3 uri paths

**Note**

This method will only update the modification time of the AWS S3 object.

**Examples**

```
## Not run:
# Require AWS S3 credentials

dir = s3_file_temp(tmp_dir = "MyBucket")
path = s3_path(dir, letters[1:2], ext = "txt")

s3_file_touch(path)

## End(Not run)
```

---

upload	<i>Upload file and directory</i>
--------	----------------------------------

---

**Description**

s3\_file\_upload upload files to AWS S3

s3\_dir\_upload upload directory to AWS S3

**Usage**

```
s3_file_upload(
  path,
  new_path,
  max_batch = fs_bytes("100MB"),
  overwrite = FALSE,
  ...
)

s3_dir_upload(path, new_path, max_batch, overwrite = FALSE, ...)
```

**Arguments**

path	(character): A character vector of local file paths to upload to AWS S3
new_path	(character): A character vector of AWS S3 paths or uri's of the new locations.
max_batch	( <a href="#">fs_bytes</a> ): Maximum batch size being uploaded with each multipart.
overwrite	(logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.
...	parameters to be passed to <a href="#">s3_put_object</a> and <a href="#">s3_create_multipart_upload</a>

**Value**

character vector of s3 uri paths

---

upload_async	<i>Upload file and directory</i>
--------------	----------------------------------

---

**Description**

s3\_file\_upload upload files to AWS S3  
s3\_dir\_upload upload directory to AWS S3

**Usage**

```
s3_file_upload_async(  
  path,  
  new_path,  
  max_batch = fs_bytes("100MB"),  
  overwrite = FALSE,  
  ...  
)  
  
s3_dir_upload_async(path, new_path, max_batch, overwrite = FALSE, ...)
```

**Arguments**

- path (character): A character vector of local file paths to upload to AWS S3
- new\_path (character): A character vector of AWS S3 paths or uri's of the new locations.
- max\_batch ([fs\\_bytes](#)): Maximum batch size being uploaded with each multipart.
- overwrite (logical): Overwrite files if the exist. If this is FALSE and the file exists an error will be thrown.
- ... parameters to be passed to [s3\\_put\\_object](#) and [s3\\_create\\_multipart\\_upload](#)

**Value**

return [future](#) object of [s3\\_file\\_upload\(\)](#) [s3\\_dir\\_upload\(\)](#)

**See Also**

[future](#) [s3\\_file\\_move\(\)](#) [s3\\_file\\_upload\(\)](#) [s3\\_dir\\_upload\(\)](#)

# Index

copy, [3](#)  
copy\_async, [4](#)  
create, [5](#)  
  
delete, [6](#)  
delete\_async, [7](#)  
download, [7](#)  
download\_async, [8](#)  
  
exists, [9](#)  
  
file\_type, [10](#)  
fs\_bytes, [3](#), [4](#), [18](#), [21](#), [23](#), [26](#), [29](#), [34](#), [38](#), [39](#),  
[41](#), [42](#)  
future, [5](#), [7](#), [9](#), [33](#), [39](#), [40](#), [42](#)  
  
info, [10](#)  
  
path, [13](#)  
path\_manipulate, [14](#)  
permission, [15](#)  
  
s3\_bucket\_chmod (permission), [15](#)  
s3\_bucket\_create (create), [5](#)  
s3\_bucket\_delete, [31](#)  
s3\_copy\_object, [21](#), [23](#), [32](#), [33](#), [40](#)  
s3\_create\_bucket, [5](#), [25](#)  
s3\_create\_multipart\_upload, [23](#), [26](#), [29](#),  
[41](#), [42](#)  
s3\_delete\_objects, [6](#), [7](#), [19](#)  
s3\_dir\_copy (copy), [3](#)  
s3\_dir\_copy(), [5](#)  
s3\_dir\_copy\_async (copy\_async), [4](#)  
s3\_dir\_create (create), [5](#)  
s3\_dir\_delete (delete), [6](#)  
s3\_dir\_delete(), [7](#)  
s3\_dir\_delete\_async (delete\_async), [7](#)  
s3\_dir\_download (download), [7](#)  
s3\_dir\_download(), [9](#)  
s3\_dir\_download\_async (download\_async),  
[8](#)  
  
s3\_dir\_exists (exists), [9](#)  
s3\_dir\_info (info), [10](#)  
s3\_dir\_ls, [28](#), [32](#)  
s3\_dir\_ls (info), [10](#)  
s3\_dir\_ls\_url, [31](#)  
s3\_dir\_tree, [32](#)  
s3\_dir\_upload (upload), [41](#)  
s3\_dir\_upload(), [42](#)  
s3\_dir\_upload\_async (upload\_async), [42](#)  
s3\_file\_chmod (permission), [15](#)  
s3\_file\_copy (copy), [3](#)  
s3\_file\_copy(), [5](#)  
s3\_file\_copy\_async (copy\_async), [4](#)  
s3\_file\_create, [40](#)  
s3\_file\_create (create), [5](#)  
s3\_file\_delete (delete), [6](#)  
s3\_file\_delete(), [7](#)  
s3\_file\_delete\_async (delete\_async), [7](#)  
s3\_file\_download (download), [7](#)  
s3\_file\_download(), [9](#)  
s3\_file\_download\_async  
(download\_async), [8](#)  
s3\_file\_exists (exists), [9](#)  
s3\_file\_info (info), [10](#)  
s3\_file\_move, [32](#)  
s3\_file\_move(), [33](#), [40](#), [42](#)  
s3\_file\_move\_async, [33](#)  
s3\_file\_size (info), [10](#)  
s3\_file\_stream\_in (stream), [38](#)  
s3\_file\_stream\_in(), [39](#), [40](#)  
s3\_file\_stream\_in\_async (stream\_async),  
[39](#)  
s3\_file\_stream\_out (stream), [38](#)  
s3\_file\_stream\_out(), [39](#), [40](#)  
s3\_file\_stream\_out\_async  
(stream\_async), [39](#)  
s3\_file\_system, [34](#)  
s3\_file\_system(), [5](#)  
s3\_file\_tag\_delete (tag), [40](#)

`s3_file_tag_info` (tag), 40  
`s3_file_tag_update` (tag), 40  
`s3_file_temp`, 35  
`s3_file_touch` (touch), 40  
`s3_file_upload` (upload), 41  
`s3_file_upload()`, 42  
`s3_file_upload_async` (upload\_async), 42  
`s3_file_url`, 36  
`s3_file_version_info`, 36  
`s3_generate_presigned_url`, 36  
`s3_get_object`, 8, 19, 21, 23, 27, 38, 39  
`s3_is_bucket` (file\_type), 10  
`s3_is_dir` (file\_type), 10  
`s3_is_file` (file\_type), 10  
`s3_is_file_empty` (file\_type), 10  
`s3_list_object_versions`, 24, 36  
`s3_list_objects_v2`, 10, 11, 24, 27, 28, 31  
`s3_path` (path), 13  
`s3_path_dir` (path\_manipulate), 14  
`s3_path_ext` (path\_manipulate), 14  
`s3_path_ext_remove` (path\_manipulate), 14  
`s3_path_ext_set` (path\_manipulate), 14  
`s3_path_file` (path\_manipulate), 14  
`s3_path_join`, 37  
`s3_path_split`, 37  
`s3_put_object`, 3–5, 18, 19, 22, 23, 26, 29, 38, 39, 41, 42  
`S3FileSystem`, 16  
`s3fs` (s3fs-package), 2  
`s3fs-package`, 2  
`stream`, 38  
`stream_async`, 39  
  
`tag`, 40  
`touch`, 40  
  
`upload`, 41  
`upload_async`, 42