

---

# **GCSFs Documentation**

*Release 0.6.0+8.g18fd9d7*

**Continuum Analytics**

**Mar 25, 2020**



---

# Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Examples</b>	<b>5</b>
<b>3</b>	<b>Credentials</b>	<b>7</b>
<b>4</b>	<b>Contents</b>	<b>9</b>
4.1	API . . . . .	9
4.2	For Developers . . . . .	15
4.3	GCSFS and FUSE . . . . .	15
4.4	Changelog . . . . .	16
4.5	Version 0.6.0 . . . . .	16
4.6	Version 0.5.3 . . . . .	16
4.7	Version 0.5.2 . . . . .	17
4.8	Version 0.5.1 . . . . .	17
4.9	Version 0.5.0 . . . . .	17
4.10	Version 0.4.0 . . . . .	17
<b>5</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



A pythonic file-system interface to [Google Cloud Storage](#).

This software is beta, use at your own risk.

Please file issues and requests on [github](#) and we welcome pull requests.

This package depends on [fsspec](#) , and inherits many useful behaviours from there, including integration with Dask, and the facility for key-value dict-like objects of the type used by zarr.



# CHAPTER 1

---

## Installation

---

The GCSFS library can be installed using conda or pip:

```
conda install -c conda-forge gcsfs  
or  
pip install gcsfs
```

or by cloning the repository:

```
git clone https://github.com/dask/gcsfs/  
cd gcsfs/  
pip install .
```



Locate and read a file:

```
>>> import gcsfs
>>> fs = gcsfs.GCSFileSystem(project='my-google-project')
>>> fs.ls('my-bucket')
['my-file.txt']
>>> with fs.open('my-bucket/my-file.txt', 'rb') as f:
...     print(f.read())
b'Hello, world'
```

(see also `walk` and `glob`)

Read with delimited blocks:

```
>>> fs.read_block(path, offset=1000, length=10, delimiter=b'\n')
b'A whole line of text\n'
```

Write with blocked caching:

```
>>> with fs.open('mybucket/new-file', 'wb') as f:
...     f.write(2*2**20 * b'a')
...     f.write(2*2**20 * b'a') # data is flushed and file closed
>>> fs.du('mybucket/new-file')
{'mybucket/new-file': 4194304}
```

Because GCSFS faithfully copies the Python file interface it can be used smoothly with other projects that consume the file interface like `gzip` or `pandas`.

```
>>> with fs.open('mybucket/my-file.csv.gz', 'rb') as f:
...     g = gzip.GzipFile(fileobj=f) # Decompress data with gzip
...     df = pd.read_csv(g)         # Read CSV file with Pandas
```



Several modes of authentication are supported:

- if `token=None` (default), GCSFS will attempt to use your default gcloud credentials or, attempt to get credentials from the google metadata service, or fall back to anonymous access. This will work for most users without further action. Note that the default project may also be found, but it is often best to supply this anyway (only affects bucket- level operations).
- if `token='cloud'`, we assume we are running within google (compute or container engine) and fetch the credentials automatically from the metadata service.
- you may supply a token generated by the `gcloud` utility; this is either a python dictionary, or the name of a file containing the JSON returned by logging in with the gcloud CLI tool (e.g., `~/.config/gcloud/application_default_credentials.json` or `~/.config/gcloud/legacy_credentials/<YOUR GOOGLE USERNAME>/adc.json`) or any value `google.Credentials` object.
- you can also generate tokens via Oauth2 in the browser using `token='browser'`, which gcsfs then caches in a special file, `~/.gcs_tokens`, and can subsequently be accessed with `token='cache'`.
- anonymous only access can be selected using `token='anon'`, e.g. to access public resources such as `'anaconda-public-data'`.

The acquired session tokens are *not* preserved when serializing the instances, so it is safe to pass them to worker processes on other machines if using in a distributed computation context. If credentials are given by a file path, however, then this file must exist on every machine.



## 4.1 API

<i>GCSFileSystem</i> ([project, access, token, ...])	Connect to Google Cloud Storage.
<i>GCSFileSystem.cat</i> (self, path)	Simple one-shot get of file data
<i>GCSFileSystem.du</i> (self, path[, total, maxdepth])	Space used by files within a path
<i>GCSFileSystem.exists</i> (self, path)	Is there a file at the given path
<i>GCSFileSystem.get</i> (self, rpath, lpath[, ...])	Copy file to local.
<i>GCSFileSystem.glob</i> (self, path, <i>\**\**kwargs</i> )	Find files by glob-matching.
<i>GCSFileSystem.info</i> (self, path, <i>\**\**kwargs</i> )	File information about this path.
<i>GCSFileSystem.ls</i> (self, path[, detail])	List objects under the given <i>'/{bucket}/{prefix}'</i> path.
<i>GCSFileSystem.mkdir</i> (self, bucket[, acl, ...])	New bucket
<i>GCSFileSystem.mv</i> (self, path1, path2, <i>\**\**kwargs</i> )	Move file from one location to another
<i>GCSFileSystem.open</i> (self, path[, mode, ...])	Return a file-like object from the filesystem
<i>GCSFileSystem.put</i> (self, lpath, rpath[, ...])	Upload file from local
<i>GCSFileSystem.read_block</i> (self, fn, offset, ...)	Read a block of bytes from
<i>GCSFileSystem.rm</i> (self, path[, recursive, ...])	Delete keys.
<i>GCSFileSystem.tail</i> (self, path[, size])	Get the last <i>size</i> bytes from file
<i>GCSFileSystem.touch</i> (self, path[, truncate])	Create empty file, or update timestamp
<i>GCSFileSystem.get_mapper</i> (self, root[, ...])	Create key/value store based on this file-system

---

*GCSFile*(gcsfs, path[, mode, block\_size, ...])

**Attributes**

<i>GCSFile.close</i> (self)	Close file
<i>GCSFile.flush</i> (self[, force])	Write buffered data to backend store.
<i>GCSFile.info</i> (self)	File information about this path
<i>GCSFile.read</i> (self[, length])	Return data from cache, or fetch pieces as necessary
<i>GCSFile.seek</i> (self, loc[, whence])	Set current file location

Continued on next page

Table 2 – continued from previous page

<code>GCSFile.tell(self)</code>	Current file location
<code>GCSFile.write(self, data)</code>	Write data to buffer.

```
class gcsfs.core.GCSFileSystem (project="", access='full_control', token=None,
                                block_size=None, consistency='none', cache_timeout=None,
                                secure_serialize=True, check_connection=False, re-
                                quests_timeout=None, requester_pays=False, **kwargs)
```

Connect to Google Cloud Storage.

The following modes of authentication are supported:

- `token=None`, GCSFS will attempt to guess your credentials in the following order: gcloud CLI default, gcsfs cached token, google compute metadata service, anonymous.
- `token='google_default'`, your default gcloud credentials will be used, which are typically established by doing `gcloud login` in a terminal.
- `token=='cache'`, credentials from previously successful gcsfs authentication will be used (use this after “browser” auth succeeded)
- `token='anon'`, no authentication is performed, and you can only access data which is accessible to allUsers (in this case, the project and access level parameters are meaningless)
- `token='browser'`, you get an access code with which you can authenticate via a specially provided URL
- if `token='cloud'`, we assume we are running within google compute or google container engine, and query the internal metadata directly for a token.
- you may supply a token generated by the `[gcloud](https://cloud.google.com/sdk/docs/)` utility; this is either a python dictionary, the name of a file containing the JSON returned by logging in with the gcloud CLI tool, or a Credentials object. gcloud typically stores its tokens in locations such as `~/.config/gcloud/application_default_credentials.json`, “`~/.config/gcloud/credentials`“, or `~\AppData\Roaming\gcloud\credentials`, etc.

Specific methods, (eg. `ls`, `info`, ...) may return object details from GCS. These detailed listings include the [object resource]([https://cloud.google.com/storage/docs/json\\_api/v1/objects#resource](https://cloud.google.com/storage/docs/json_api/v1/objects#resource))

GCS *does not* include “directory” objects but instead generates directories by splitting [object names](<https://cloud.google.com/storage/docs/key-terms>). This means that, for example, a directory does not need to exist for an object to be created within it. Creating an object implicitly creates it’s parent directories, and removing all objects from a directory implicitly deletes the empty directory.

`GCSFileSystem` generates listing entries for these implied directories in listing apis with the object properties:

- **“name”** [string] The “{bucket}/{name}” path of the dir, used in calls to `GCSFileSystem` or `GCSFile`.
- **“bucket”** [string] The name of the bucket containing this object.
- **“kind”** : ‘storage#object’
- **“size”** : 0
- **“storageClass”** : ‘DIRECTORY’
- **type**: ‘directory’ (fsspec compat)

`GCSFileSystem` maintains a per-implied-directory cache of object listings and fulfills all object information and listing requests from cache. This implied, for example, that objects created via other processes *will not* be visible to the `GCSFileSystem` until the cache refreshed. Calls to `GCSFileSystem.open` and calls to `GCSFile` are not effected by this cache.

In the default case the cache is never expired. This may be controlled via the `cache_timeout` `GCSFileSystem` parameter or via explicit calls to `GCSFileSystem.invalidate_cache`.

### Parameters

- project** [string] `project_id` to work under. Note that this is not the same as, but often very similar to, the project name. This is required in order to list all the buckets you have access to within a project and to create/delete buckets, or update their access policies. If `token='google_default'`, the value is overridden by the default, if `token='anon'`, the value is ignored.
- access** [one of {'read\_only', 'read\_write', 'full\_control'}] Full control implies read/write as well as modifying metadata, e.g., access control.
- token: None, dict or string** (see description of authentication methods, above)
- consistency: 'none', 'size', 'md5'** Check method when writing files. Can be overridden in `open()`.
- cache\_timeout: float, seconds** Cache expiration time in seconds for object metadata cache. Set `cache_timeout <= 0` for no caching, `None` for no cache expiration.
- secure\_serialize: bool** If `True`, instances re-establish auth upon deserialization; if `False`, token is passed directly, which may be a security risk if passed across an insecure network.
- check\_connection: bool** When `token=None`, `gcsfs` will attempt various methods of establishing credentials, falling back to `anon`. It is possible for a method to find credentials in the system that turn out not to be valid. Setting this parameter to `True` will ensure that an actual operation is attempted before deciding that credentials are valid.
- requester\_pays** [bool, or str default `False`] Whether to use requester-pays requests. This will include your project ID `project` in requests as the `userProject`, and you'll be billed for accessing data from requester-pays buckets. Optionally, pass a project-id here as a string to use that as the `userProject`.

### Attributes

- buckets** Return list of available project buckets.
- transaction** A context within which files are committed together upon exit

### Methods

<code>cat(self, path)</code>	Simple one-shot get of file data
<code>checksum(self, path)</code>	Unique value for current version of file
<code>clear_instance_cache()</code>	Clear the cache of filesystem instances.
<code>connect(self[, method])</code>	Establish session token.
<code>copy(self, path1, path2[, acl])</code>	Duplicate remote file
<code>cp(self, path1, path2, **kwargs)</code>	Alias of <code>FilesystemSpec.copy</code> .
<code>current()</code>	Return the most recently created <code>FileSystem</code>
<code>delete(self, path[, recursive, maxdepth])</code>	Alias of <code>FilesystemSpec.rm</code> .
<code>disk_usage(self, path[, total, maxdepth])</code>	Alias of <code>FilesystemSpec.du</code> .
<code>download(self, rpath, lpath[, recursive])</code>	Alias of <code>FilesystemSpec.get</code> .
<code>du(self, path[, total, maxdepth])</code>	Space used by files within a path
<code>end_transaction(self)</code>	Finish write transaction, non-context version
<code>exists(self, path)</code>	Is there a file at the given path
<code>find(self, path[, maxdepth, withdirs])</code>	List all files below path.

Continued on next page

Table 3 – continued from previous page

<code>from_json(blob)</code>	Recreate a filesystem instance from JSON representation
<code>get(self, rpath, lpath[, recursive])</code>	Copy file to local.
<code>get_mapper(self, root[, check, create])</code>	Create key/value store based on this file-system
<code>getxattr(self, path, attr)</code>	Get user-defined metadata attribute
<code>glob(self, path, \**kwargs)</code>	Find files by glob-matching.
<code>head(self, path[, size])</code>	Get the first <code>size</code> bytes from file
<code>info(self, path, \**kwargs)</code>	File information about this path.
<code>invalidate_cache(self[, path])</code>	Invalidate listing cache for given path, it is reloaded on next use.
<code>isdir(self, path)</code>	Is this entry directory-like?
<code>isfile(self, path)</code>	Is this entry file-like?
<code>listdir(self, path[, detail])</code>	Alias of <code>FilesystemSpec.ls</code> .
<code>load_tokens()</code>	Get “browser” tokens from disc
<code>ls(self, path[, detail])</code>	List objects under the given <code>'/{bucket}/{prefix}</code> path.
<code>makedirs(self, path[, create_parents])</code>	Alias of <code>FilesystemSpec.mkdir</code> .
<code>makedirs(self, path[, exist_ok])</code>	Recursively make directories
<code>merge(self, path, paths[, acl])</code>	Concatenate objects within a single bucket
<code>mkdir(self, bucket[, acl, default_acl])</code>	New bucket
<code>makedirs(self, path[, exist_ok])</code>	Alias of <code>FilesystemSpec.makedirs</code> .
<code>move(self, path1, path2, \**kwargs)</code>	Alias of <code>FilesystemSpec.mv</code> .
<code>mv(self, path1, path2, \**kwargs)</code>	Move file from one location to another
<code>open(self, path[, mode, block_size, ...])</code>	Return a file-like object from the filesystem
<code>put(self, lpath, rpath[, recursive])</code>	Upload file from local
<code>read_block(self, fn, offset, length[, delimiter])</code>	Read a block of bytes from
<code>rename(self, path1, path2, \**kwargs)</code>	Alias of <code>FilesystemSpec.mv</code> .
<code>rm(self, path[, recursive, maxdepth])</code>	Delete keys.
<code>rmdir(self, bucket)</code>	Delete an empty bucket
<code>setxattrs(self, path[, content_type, ...])</code>	Set/delete/add writable metadata attributes
<code>size(self, path)</code>	Size in bytes of file
<code>split_path(path)</code>	Normalise GCS path string into bucket and key.
<code>start_transaction(self)</code>	Begin write transaction for deferring files, non-context version
<code>stat(self, path, \**kwargs)</code>	Alias of <code>FilesystemSpec.info</code> .
<code>tail(self, path[, size])</code>	Get the last <code>size</code> bytes from file
<code>to_json(self)</code>	JSON representation of this filesystem instance
<code>touch(self, path[, truncate])</code>	Create empty file, or update timestamp
<code>ukey(self, path)</code>	Hash of file properties, to tell if it has changed
<code>upload(self, lpath, rpath[, recursive])</code>	Alias of <code>FilesystemSpec.put</code> .
<code>url(path)</code>	Get HTTP URL of the given path
<code>walk(self, path[, maxdepth])</code>	Return all files belows path

**buckets**

Return list of available project buckets.

**cat** (*self, path*)

Simple one-shot get of file data

**connect** (*self, method=None*)

Establish session token. A new token will be requested if the current one is within 100s of expiry.

**Parameters**

**method:** `str` (`google_default|cache|cloud|token|anon|browser`) or `None` Type of authori-

sation to implement - calls `_connect_*` methods. If None, will try sequence of methods.

**copy** (*self*, *path1*, *path2*, *acl=None*)  
Duplicate remote file

**getxattr** (*self*, *path*, *attr*)  
Get user-defined metadata attribute

**info** (*self*, *path*, *\*\*kwargs*)  
File information about this path.

**invalidate\_cache** (*self*, *path=None*)  
Invalidate listing cache for given path, it is reloaded on next use.

#### Parameters

**path: string or None** If None, clear all listings cached else listings at or under given path.

**classmethod load\_tokens** ()  
Get “browser” tokens from disc

**ls** (*self*, *path*, *detail=False*, *\*\*kwargs*)  
List objects under the given ‘/{bucket}/{prefix}’ path.

**merge** (*self*, *path*, *paths*, *acl=None*)  
Concatenate objects within a single bucket

**mkdir** (*self*, *bucket*, *acl='projectPrivate'*, *default\_acl='bucketOwnerFullControl'*)  
New bucket

#### Parameters

**bucket: str** bucket name. If contains ‘/’ (i.e., looks like subdir), will have no effect because GCS doesn’t have real directories.

**acl: string, one of bACLs** access for the bucket itself

**default\_acl: str, one of ACLs** default ACL for objects created in this bucket

**rm** (*self*, *path*, *recursive=False*, *maxdepth=None*)  
Delete keys.

If a list, batch-delete all keys in one go (can span buckets)

Returns whether operation succeeded (a list if input was a list)

If recursive, delete all keys given by find(path)

**rmdir** (*self*, *bucket*)  
Delete an empty bucket

#### Parameters

**bucket: str** bucket name. If contains ‘/’ (i.e., looks like subdir), will have no effect because GCS doesn’t have real directories.

**setxattrs** (*self*, *path*, *content\_type=None*, *content\_encoding=None*, *\*\*kwargs*)  
Set/delete/add writable metadata attributes

**content\_type: str** If not None, set the content-type to this value

**content\_encoding: str** If not None, set the content-encoding. See <https://cloud.google.com/storage/docs/transcoding>

**kw\_args: key-value pairs like field="value" or field=None** value must be string to add or modify, or None to delete

**Returns****Entire metadata after update (even if only path is passed)****classmethod** `split_path` (*path*)

Normalise GCS path string into bucket and key.

**Parameters****path** [string] Input path, like `gcs://mybucket/path/to/file`. Path is of the form: `'[gs|gcs://]bucket[/key]'`**Returns****(bucket, key) tuple****classmethod** `url` (*path*)

Get HTTP URL of the given path

**class** `gcsfs.core.GCSFile` (*gcsfs, path, mode='rb', block\_size=5242880, autoccommit=True, cache\_type='readahead', cache\_options=None, acl=None, consistency='md5', metadata=None, content\_type=None, \*\*kwargs*)**Attributes****closed****Methods**

<code>close(self)</code>	Close file
<code>commit(self)</code>	If not auto-committing, finalize file
<code>discard(self)</code>	Cancel in-progress multi-upload
<code>fileno(self, /)</code>	Returns underlying file descriptor if one exists.
<code>flush(self[, force])</code>	Write buffered data to backend store.
<code>info(self)</code>	File information about this path
<code>isatty(self, /)</code>	Return whether this is an 'interactive' stream.
<code>read(self[, length])</code>	Return data from cache, or fetch pieces as necessary
<code>readable(self)</code>	Whether opened for reading
<code>readinto(self, b)</code>	mirrors builtin file's readinto method
<code>readline(self)</code>	Read until first occurrence of newline character
<code>readlines(self)</code>	Return all data, split by the newline character
<code>readuntil(self[, char, blocks])</code>	Return data between current position and first occurrence of char
<code>seek(self, loc[, whence])</code>	Set current file location
<code>seekable(self)</code>	Whether is seekable (only in read mode)
<code>tell(self)</code>	Current file location
<code>truncate()</code>	Truncate file to size bytes.
<code>url(self)</code>	HTTP link to this file's data
<code>writable(self)</code>	Whether opened for writing
<code>write(self, data)</code>	Write data to buffer.

<b>readinto1</b>	
<b>writelines</b>	

**commit** (*self*)

If not auto-committing, finalize file

**discard** (*self*)

Cancel in-progress multi-upload

Should only happen during discarding this write-mode file

**info** (*self*)

File information about this path

**url** (*self*)

HTTP link to this file's data

## 4.2 For Developers

We welcome contributions to gcsfs!

Please file issues and requests on [github](#) and we welcome pull requests.

### 4.2.1 Testing and VCR

VCR records requests to the remote server, so that they can be replayed during tests - so long as the requests match exactly the original. It is set to strip out sensitive information before writing the request and responses into yaml files in the tests/recordings/ directory; the file-name matches the test name, so all tests must have unique names, across all test files.

The process is as follows:

- Create a bucket for testing
- Set environment variables so that the tests run against your GCS credentials, and recording occurs

```
export GCSFS_RECORD_MODE=all
export GCSFS_TEST_PROJECT='...'
export GCSFS_TEST_BUCKET='...' # the bucket from step 1.
export GCSFS_GOOGLE_TOKEN=~/.config/gcloud/application_default_credentials.json
py.test -vv -x -s gcsfs
```

These variables can also be set in gcsfs/tests/settings.py

- Run this again, setting GCSFS\_RECORD\_MODE=once, which should alert you if your tests make different requests the second time around
- Finally, test as TravisCI will, using only the recordings

```
export GCSFS_RECORD_MODE=none
py.test -vv -x -s gcsfs
```

To reset recording and start again, delete the yaml file corresponding to the test in gcsfs/tests/recordings/\*.yaml.

## 4.3 GCSFS and FUSE

Warning, this functionality is **experimental**

FUSE is a mechanism to mount user-level filesystems in unix-like systems (linux, osx, etc.). GCSFS is able to use FUSE to present remote data/keys as if they were a directory on your local file-system. This allows for standard shell command manipulation, and loading of data by libraries that can only handle local file-paths (e.g., netCDF/HDF5).

### 4.3.1 Requirements

In addition to a standard installation of GCSFS, you also need:

- libfuse as a system install. The way to install this will depend on your OS. Examples include `sudo apt-get install fuse`, `sudo yum install fuse` and download from [osxfuse](#).
- `fusepy`, which can be installed via conda or pip
- `pandas`, which can also be installed via conda or pip (this library is used only for its timestring parsing).

### 4.3.2 Usage

FUSE functionality is available via the `fsspec.fuse` module. See the docstrings for further details.

```
gcs = gcsfs.GCSFileSystem(..)
from fsspec.fuse import run
run(gcs, "bucket/path", "local/path", foreground=True, threads=False)
```

### 4.3.3 Caveats

This functionality is experimental. The command usage may change, and you should expect exceptions.

Furthermore:

- although mutation operations tentatively work, you should not at the moment depend on `gcsfuse` as a reliable system that won't lose your data.
- permissions on GCS are complicated, so all files will be shown as fully-open `0o777`, regardless of state. If a read fails, you likely don't have the right permissions.

## 4.4 Changelog

### 4.5 Version 0.6.0

- **API-breaking:** Changed requester-pays handling for `GCSFileSystem`.

The `user_project` keyword has been removed, and has been replaced with the `requester_pays` keyword. If you're working with a `requester_pays` bucket you will need to explicitly pass `requester_pays=True`. This will include your `project` ID in requests made to GCS.

### 4.6 Version 0.5.3

- `GCSFileSystem` now validates that the `project` provided, if any, matches the Google default project when using `token='google_default'` to authenticate (PR #219).
- Fixed bug in `GCSFileSystem.cat` on objects in requester-pays buckets (PR #217).

## 4.7 Version 0.5.2

- Fixed bug in `user_project` fallback for default Google authentication (PR #213)

## 4.8 Version 0.5.1

- `user_project` now falls back to the `project` if provided (PR #208)

## 4.9 Version 0.5.0

- Added the ability to make requester-pays requests with the `user_project` parameter (PR #206)

## 4.10 Version 0.4.0

- Improved performance when serializing filesystem objects (PR #182)
- Fixed authorization errors when using `gcsfs` within multithreaded code (PR #183, PR #192)
- Added contributing instructions (PR #185)
- Improved performance for `gcsfs.GCSFileSystem.info()` (PR #187)
- Fixed bug in `gcsfs.GCSFileSystem.info()` raising an error (PR #190)



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



## B

buckets (*gcsfs.core.GCSFileSystem* attribute), 12

## C

cat () (*gcsfs.core.GCSFileSystem* method), 12

commit () (*gcsfs.core.GCSFile* method), 14

connect () (*gcsfs.core.GCSFileSystem* method), 12

copy () (*gcsfs.core.GCSFileSystem* method), 13

## D

discard () (*gcsfs.core.GCSFile* method), 15

## G

GCSFile (*class in gcsfs.core*), 14

GCSFileSystem (*class in gcsfs.core*), 10

getxattr () (*gcsfs.core.GCSFileSystem* method), 13

## I

info () (*gcsfs.core.GCSFile* method), 15

info () (*gcsfs.core.GCSFileSystem* method), 13

invalidate\_cache () (*gcsfs.core.GCSFileSystem* method), 13

## L

load\_tokens () (*gcsfs.core.GCSFileSystem* class method), 13

ls () (*gcsfs.core.GCSFileSystem* method), 13

## M

merge () (*gcsfs.core.GCSFileSystem* method), 13

mkdir () (*gcsfs.core.GCSFileSystem* method), 13

## R

rm () (*gcsfs.core.GCSFileSystem* method), 13

rmdir () (*gcsfs.core.GCSFileSystem* method), 13

## S

setxattrs () (*gcsfs.core.GCSFileSystem* method), 13

split\_path () (*gcsfs.core.GCSFileSystem* class method), 14

## U

url () (*gcsfs.core.GCSFile* method), 15

url () (*gcsfs.core.GCSFileSystem* class method), 14