GCSFs Documentation

Release 2022.11.0

Continuum Analytics

Nov 23, 2022

CONTENTS

1 Installation **3**

2 Examples **5**

3 Credentials **7**

4 Integration **9**

5 Async **11**

6 Proxy **13**

7 Contents **15**
- 7.1 API . . . 15
- 7.2 For Developers . . . 22
- 7.3 GCSFS and FUSE . . . 23
- 7.4 Changelog . . . 24

8 Indices and tables **29**

Index **31**

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

ONE

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/fsspec/gcsfs/
cd gcsfs/
pip install .
```

CHAPTER

TWO

EXAMPLES

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
```

CHAPTER

THREE

CREDENTIALS

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.

CHAPTER

FOUR

INTEGRATION

The libraries `intake`, `pandas` and `dask` accept URLs with the prefix "gcs://", and will use gcsfs to complete the IO operation in question. The IO functions take an argument `storage_options`, which will be passed to `GCSFileSystem`, for example:

```
df = pd.read_excel("gcs://bucket/path/file.xls",
                   storage_options={"token": "anon"})
```

This gives the chance to pass any credentials or other necessary arguments needed to s3fs.

CHAPTER

FIVE

ASYNC

`gcsfs` is implemented using `aiohttp`, and offers async functionality. A number of methods of `GCSFileSystem` are `async`, for for each of these, there is also a synchronous version with the same name and lack of a "_" prefix.

If you wish to call `gcsfs` from async code, then you should pass `asynchronous=True, loop=loop` to the constructor (the latter is optional, if you wish to use both async and sync methods). You must also explicitly await the client creation before making any GCS call.

```
loop = ...  # however you create your loop

async def run_program(loop):
    gcs = GCSFileSystem(..., asynchronous=True, loop=loop)
    await gcs.set_session()
    ...  # perform work

asyncio.run(run_program(loop))  # or call from your async code
```

Concurrent async operations are also used internally for bulk operations such as `pipe/cat`, `get/put`, `cp/mv/rm`. The async calls are hidden behind a synchronisation layer, so are designed to be called from normal code. If you are *not* using async-style programming, you do not need to know about how this works, but you might find the implementation interesting.

CHAPTER

SIX

PROXY

`gcsfs` uses `aiohttp` for calls to the storage api, which by default ignores `HTTP_PROXY/HTTPS_PROXY` environment variables. To read proxy settings from the environment provide `session_kwargs` as follows:

```
fs = GCSFileSystem(project='my-google-project', session_kwargs={'trust_env': True})
```

For further reference check aiohttp proxy support.

CHAPTER

SEVEN

CONTENTS

7.1 API

GCSFileSystem(*args, **kwargs)	Connect to Google Cloud Storage.
GCSFileSystem.cat(path[, recursive, on_error])	Fetch (potentially multiple) paths' contents
GCSFileSystem.du(path[, total, maxdepth])	Space used by files within a path
GCSFileSystem.exists(path, **kwargs)	Is there a file at the given path
GCSFileSystem.get(rpath, lpath[, recursive, ...])	Copy file(s) to local.
GCSFileSystem.glob(path, **kwargs)	Find files by glob-matching.
GCSFileSystem.info(path, **kwargs)	Give details of entry at path
GCSFileSystem.ls(path[, detail])	List objects at path.
GCSFileSystem.mkdir(path[, acl, ...])	New bucket
GCSFileSystem.mv(path1, path2[, recursive, ...])	Move file(s) from one location to another
GCSFileSystem.open(path[, mode, block_size, ...])	Return a file-like object from the filesystem
GCSFileSystem.put(lpath, rpath[, recursive, ...])	Copy file(s) from local.
GCSFileSystem.read_block(fn, offset, length)	Read a block of bytes from
GCSFileSystem.rm(path[, recursive, ...])	Delete files.
GCSFileSystem.tail(path[, size])	Get the last size bytes from file
GCSFileSystem.touch(path[, truncate])	Create empty file, or update timestamp
GCSFileSystem.get_mapper([root, check, ...])	Create key/value store based on this file-system

GCSFile(gcsfs, path[, mode, block_size, ...])	
Attributes	
GCSFile.close()	Close file
GCSFile.flush([force])	Write buffered data to backend store.
GCSFile.info()	File information about this path
GCSFile.read([length])	Return data from cache, or fetch pieces as necessary
GCSFile.seek(loc[, whence])	Set current file location
GCSFile.tell()	Current file location
GCSFile.write(data)	Write data to buffer.

class gcsfs.core.**GCSFileSystem**(**args*, ***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)

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 (deprecated)
requester_pays
[bool, or str default False] Whether to use requester-pays requests. This will include your project ID *project* in requests as the *userPorject*, 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*.

session_kwargs: dict
passed on to aiohttp.ClientSession; can contain, for example, proxy settings.

endpoint_url: str
If given, use this URL (format protocol://host:port , *without* any path part) for communication. If not given, defaults to the value of environment variable "STORAGE_EMULATOR_HOST"; if that is not set either, will use the standard Google endpoint.

default_location: str
Default location where buckets are created, like 'US' or 'EUROPE-WEST3'. You can find a list of all available locations here: https://cloud.google.com/storage/docs/locations#available-locations

version_aware: bool
Whether to support object versioning. If enabled this will require the user to have the necessary permissions for dealing with versioned objects.

Attributes

base
buckets
Return list of available project buckets.

loop
on_google
project
session
`transaction`
A context within which files are committed together upon exit

Methods

cat(path[, recursive, on_error])	Fetch (potentially multiple) paths' contents
cat_file(path[, start, end])	Get the content of a file
checksum(path)	Unique value for current version of file
clear_instance_cache()	Clear the cache of filesystem instances.
copy(path1, path2[, recursive, on_error])	Copy within two locations in the filesystem
cp(path1, path2, **kwargs)	Alias of *AbstractFileSystem.copy*.
created(path)	Return the created timestamp of a file as a datetime.datetime
current()	Return the most recently instantiated FileSystem
delete(path[, recursive, maxdepth])	Alias of *AbstractFileSystem.rm*.
disk_usage(path[, total, maxdepth])	Alias of *AbstractFileSystem.du*.
download(rpath, lpath[, recursive])	Alias of *AbstractFileSystem.get*.
du(path[, total, maxdepth])	Space used by files within a path
end_transaction()	Finish write transaction, non-context version
exists(path, **kwargs)	Is there a file at the given path
expand_path(path[, recursive, maxdepth])	Turn one or more globs or directories into a list of all matching paths to files or directories.
find(path[, maxdepth, withdirs, detail])	List all files below path.
from_json(blob)	Recreate a filesystem instance from JSON representation
get(rpath, lpath[, recursive, callback])	Copy file(s) to local.
get_file(rpath, lpath[, callback, outfile])	Copy single remote file to local
get_mapper([root, check, create, ...])	Create key/value store based on this file-system
getxattr(path, attr)	Get user-defined metadata attribute
glob(path, **kwargs)	Find files by glob-matching.
head(path[, size])	Get the first size bytes from file
info(path, **kwargs)	Give details of entry at path
invalidate_cache([path])	Invalidate listing cache for given path, it is reloaded on next use.
isdir(path)	Is this entry directory-like?
isfile(path)	Is this entry file-like?
lexists(path, **kwargs)	If there is a file at the given path (including broken links)
listdir(path[, detail])	Alias of *AbstractFileSystem.ls*.
ls(path[, detail])	List objects at path.
makedir(path[, create_parents])	Alias of *AbstractFileSystem.mkdir*.
makedirs(path[, exist_ok])	Recursively make directories
merge(path, paths[, acl])	Concatenate objects within a single bucket
mkdir(path[, acl, default_acl, location, ...])	New bucket
mkdirs(path[, exist_ok])	Alias of *AbstractFileSystem.makedirs*.
modified(path)	Return the modified timestamp of a file as a datetime.datetime
move(path1, path2, **kwargs)	Alias of *AbstractFileSystem.mv*.
mv(path1, path2[, recursive, maxdepth])	Move file(s) from one location to another
open(path[, mode, block_size, ...])	Return a file-like object from the filesystem
pipe(path[, value])	Put value into path
pipe_file(path, value, **kwargs)	Set the bytes of given file
put(lpath, rpath[, recursive, callback])	Copy file(s) from local.
put_file(lpath, rpath[, callback])	Copy single file to remote

continues on next page

Table 1 – continued from previous page

read_block(fn, offset, length[, delimiter])	Read a block of bytes from
read_bytes(path[, start, end])	Alias of *AbstractFileSystem.cat_file*.
read_text(path[, encoding, errors, newline])	Get the contents of the file as a string.
rename(path1, path2, **kwargs)	Alias of *AbstractFileSystem.mv*.
rm_file(path)	Delete a file
rmdir(bucket)	Delete an empty bucket
setxattrs(path[, content_type, ...])	Set/delete/add writable metadata attributes
sign(path[, expiration])	Create a signed URL representing the given path.
size(path)	Size in bytes of file
sizes(paths)	Size in bytes of each file in a list of paths
start_transaction()	Begin write transaction for deferring files, non-context version
stat(path, **kwargs)	Alias of *AbstractFileSystem.info*.
tail(path[, size])	Get the last size bytes from file
to_json()	JSON representation of this filesystem instance
touch(path[, truncate])	Create empty file, or update timestamp
ukey(path)	Hash of file properties, to tell if it has changed
unstrip_protocol(name)	Format FS-specific path to generic, including protocol
upload(lpath, rpath[, recursive])	Alias of *AbstractFileSystem.put*.
url(path)	Get HTTP URL of the given path
walk(path[, maxdepth, topdown])	Return all files belows path
write_bytes(path, value, **kwargs)	Alias of *AbstractFileSystem.pipe_file*.
write_text(path, value[, encoding, errors, ...])	Write the text to the given file.

call	
cat_ranges	
close_session	
cp_file	
make_bucket_requester_pays	
open_async	
rm	
split_path	

property buckets

Return list of available project buckets.

getxattr(*path*, *attr*)

Get user-defined metadata attribute

invalidate_cache(*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.

merge(*path*, *paths*, *acl=None*)

Concatenate objects within a single bucket

mkdir(*path*, *acl='projectPrivate'*, *default_acl='bucketOwnerFullControl'*, *location=None*, *create_parents=True*, *enable_versioning=False*, ***kwargs*)

New bucket

If path is more than just a bucket, will create bucket if create_parents=True; otherwise is a noop. If create_parents is False and bucket does not exist, will produce FileNotFFoundError.

Parameters

path: 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

location: Optional[str]
Location where buckets are created, like 'US' or 'EUROPE-WEST3'. If not provided, defaults to *self.default_location*. You can find a list of all available locations here: https://cloud.google.com/storage/docs/locations#available-locations

create_parents: bool
If True, creates the bucket in question, if it doesn't already exist

enable_versioning: bool
If True, creates the bucket in question with object versioning enabled.

`rm`(*path*, *recursive=False*, *maxdepth=None*, *batchsize=20*)

Delete files.

Parameters

path: str or list of str
File(s) to delete.

recursive: bool
If file(s) are directories, recursively delete contents and then also remove the directory

maxdepth: int or None
Depth to pass to walk for finding files to delete, if recursive. If None, there will be no limit and infinite recursion may be possible.

`rmdir`(*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`(*path*, *content_type=None*, *content_encoding=None*, *fixed_key_metadata=None*, ***kwargs*)

Set/delete/add writable metadata attributes

Note: uses PATCH method (update), leaving unedited keys alone. fake-gcs-server:latest does not seem to support this.

7.1.1 Parameters

content_type: str
If not None, set the content-type to this value

content_encoding: str
This parameter is deprecated, you may use fixed_key_metadata instead. If not None, set the content-encoding. See https://cloud.google.com/storage/docs/transcoding

fixed_key_metadata: dict

Google metadata, in key/value pairs, supported keys:

- cache_control
- content_disposition
- content_encoding
- content_language
- custom_time

More info: https://cloud.google.com/storage/docs/metadata#mutable

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)

`sign`(*path*, *expiration=100*, ***kwargs*)

Create a signed URL representing the given path.

Parameters

path
[str] The path on the filesystem

expiration
[int] Number of seconds to enable the URL for

Returns

URL
[str] The signed URL

`url`(*path*)

Get HTTP URL of the given path

`class gcsfs.core.GCSFile`(*gcsfs*, *path*, *mode='rb'*, *block_size=5242880*, *autocommit=True*, *cache_type='readahead'*, *cache_options=None*, *acl=None*, *consistency='md5'*, *metadata=None*, *content_type=None*, *timeout=None*, *fixed_key_metadata=None*, *generation=None*, ***kwargs*)

Attributes

closed
details
full_name

Methods

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

readinto1	

commit()

If not auto-committing, finalize file

discard()

Cancel in-progress multi-upload

Should only happen during discarding this write-mode file

info()

File information about this path

url()

HTTP link to this file's data

7.2 For Developers

We welcome contributions to gcsfs!

Please file issues and requests on github and we welcome pull requests.

7.2.1 Testing

The testing framework supports using your own GCS-compliant endpoint, by setting the "STORAGE_EMULATOR_HOST" environment variable. If this is not set, then an emulator will be spun up using `docker` and fake-gcs-server. This emulator has almost all the functionality of real GCS. A small number of tests run differently or are skipped.

If you want to actually test against real GCS, then you should set STORAGE_EMULATOR_HOST to "https://storage.googleapis.com" and also provide appropriate GCSFS_TEST_BUCKET and GCSFS_TEST_PROJECT, as well as setting your default google credentials (or providing them via the fsspec config).

7.3 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).

7.3.1 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.

7.3.2 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)
```

7.3.3 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 loose 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.

7.4 Changelog

7.4.1 2022.11.0

- implement object versioning (#504)

7.4.2 2022.10.0

- bump fsspec to 2022.10.0 (#503)

7.4.3 2022.8.1

- don't install prerelease aiohttp (#490)

7.4.4 2022.7.1

- Try cloud auth by default (#479)

7.4.5 2022.5.0

- invalidate listings cache for simple put/pipe (#474)
- conform _mkdir and _cat_file to upstream (#471)

7.4.6 2022.3.0

(note that this release happened in 2022.4, but we label as 2022.3 to match fsspec)

- bucket exists workaround (#464)
- dirmarkers (#459)
- check connection (#457)
- browser connection now uses local server (#456)
- bucket location (#455)
- ensure auth is closed (#452)

7.4.7 2022.02.0

- fix list_buckets without cache (#449)
- drop py36 (#445)

7.4.8 2022.01.0

- update refname for versions (#442)

7.4.9 2021.11.1

- don't touch cache when doing find with a prefix (#437)

7.4.10 2021.11.0

- move to fsspec org
- add support for google fixed_key_metadata (#429)
- deprecate *content_encoding* parameter of setxattrs method (#429)
- use emulator for resting instead of vcrpy (#424)

7.4.11 2021.10.1

- url signing (#411)
- default callback (#422)

7.4.12 2021.10.0

- min version for decorator
- default callback in get (#422)

7.4.13 2021.09.0

- correctly recognise 404 (#419)
- fix for .details due to upstream (#417)
- callbacks in get/put (#416)
- "%" in paths (#415)

7.4.14 2021.08.1

- don't retry 404s (#406)

7.4.15 2021.07.0

- fix find/glob with a prefix (#399)

7.4.16 2021.06.1

- kwargs to aiohttpClient session
- graceful timeout when disconnecting at finalise (#397)

7.4.17 2021.06.0

- negative ranges in cat_file (#394)

7.4.18 2021.05.0

- no credentials bug fix (#390)
- use googleapis.com (#388)
- more retries (#387, 385, 380)
- Code cleanup (#381)
- license to match stated one (#378)
- deps updated (#376)

7.4.19 Version 2021.04.0

- switch to calver and fsspec pin

7.4.20 Version 0.8.0

- keep up with fsspec 0.9.0 async
- one-shot find
- consistency checkers
- retries for intermittent issues
- timeouts
- partial cat
- http error status
- CI to GHA

7.4.21 Version 0.7.0

- async operations via aiohttp

7.4.22 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.

7.4.23 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).

7.4.24 Version 0.5.2

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

7.4.25 Version 0.5.1

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

7.4.26 Version 0.5.0

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

7.4.27 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

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

B

buckets (*gcsfs.core.GCSFileSystem property*), 19

C

commit() (*gcsfs.core.GCSFile method*), 22

D

discard() (*gcsfs.core.GCSFile method*), 22

G

GCSFile (*class in gcsfs.core*), 21
GCSFileSystem (*class in gcsfs.core*), 15
getxattr() (*gcsfs.core.GCSFileSystem method*), 19

I

info() (*gcsfs.core.GCSFile method*), 22
invalidate_cache() (*gcsfs.core.GCSFileSystem method*), 19

M

merge() (*gcsfs.core.GCSFileSystem method*), 19
mkdir() (*gcsfs.core.GCSFileSystem method*), 19

R

rm() (*gcsfs.core.GCSFileSystem method*), 20
rmdir() (*gcsfs.core.GCSFileSystem method*), 20

S

setxattrs() (*gcsfs.core.GCSFileSystem method*), 20
sign() (*gcsfs.core.GCSFileSystem method*), 21

U

url() (*gcsfs.core.GCSFile method*), 22
url() (*gcsfs.core.GCSFileSystem method*), 21