GCSFs Documentation

Release 2023.6.0+1.g7cc53d9

Continuum Analytics

Jul 21, 2023

CONTENTS

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`:

```
conda install -c conda-forge gcsfs
```

or `pip`:

```
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 gcsfs.

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.

```
async def run_program():
    gcs = GCSFileSystem(asynchronous=True)
    print(await gcs._ls(""))

asyncio.run(run_program())  # 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.

For every synchronous function there is asynchronous one prefixed by _, but the `open` operation does not support async operation. If you need it to open some file in async manner, it's better to asynchronously download it to temporary location and working with it from there.

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 and optionally directories 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[, maxdepth])	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, ...])	
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 *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*.
- **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.

property buckets

Return list of available project buckets.

cat(*path*, *recursive=False*, *on_error='raise'*, ***kwargs*)

Fetch (potentially multiple) paths' contents

Parameters

- **recursive** (*bool*) – If True, assume the path(s) are directories, and get all the contained files
- **on_error** (*"raise", "omit", "return"*) – If raise, an underlying exception will be raised (converted to KeyError if the type is in self.missing_exceptions); if omit, keys with exception will simply not be included in the output; if "return", all keys are included in the output, but the value will be bytes or an exception instance.
- **kwargs** (*passed to cat_file*) –

Returns

- **dict of {path** (*contents} if there are multiple paths*)
- *or the path has been otherwise expanded*

cat_file(*path*, *start=None*, *end=None*, ***kwargs*)

Get the content of a file

Parameters

- **path** (*URL of file on this filesystems*) –
- **start** (*int*) – Bytes limits of the read. If negative, backwards from end, like usual python slices. Either can be None for start or end of file, respectively
- **end** (*int*) – Bytes limits of the read. If negative, backwards from end, like usual python slices. Either can be None for start or end of file, respectively

- **kwargs** (passed to open().) –

cat_ranges(*paths*, *starts*, *ends*, *max_gap=None*, *on_error='return'*, ***kwargs*)

Get the contents of byte ranges from one or more files

Parameters

- **paths** (*list*) – A list of of filepaths on this filesystems
- **starts** (*int or list*) – Bytes limits of the read. If using a single int, the same value will be used to read all the specified files.
- **ends** (*int or list*) – Bytes limits of the read. If using a single int, the same value will be used to read all the specified files.

checksum(*path*)

Unique value for current version of file

If the checksum is the same from one moment to another, the contents are guaranteed to be the same. If the checksum changes, the contents *might* have changed.

This should normally be overridden; default will probably capture creation/modification timestamp (which would be good) or maybe access timestamp (which would be bad)

classmethod clear_instance_cache()

Clear the cache of filesystem instances.

Notes

Unless overridden by setting the cachable class attribute to False, the filesystem class stores a reference to newly created instances. This prevents Python's normal rules around garbage collection from working, since the instances refcount will not drop to zero until clear_instance_cache is called.

copy(*path1*, *path2*, *recursive=False*, *maxdepth=None*, *on_error=None*, ***kwargs*)

Copy within two locations in the filesystem

on_error
["raise", "ignore"] If raise, any not-found exceptions will be raised; if ignore any not-found exceptions will cause the path to be skipped; defaults to raise unless recursive is true, where the default is ignore

cp(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.copy*.

created(*path*)

Return the created timestamp of a file as a datetime.datetime

classmethod current()

Return the most recently instantiated FileSystem

If no instance has been created, then create one with defaults

delete(*path*, *recursive=False*, *maxdepth=None*)

Alias of *AbstractFileSystem.rm*.

disk_usage(*path*, *total=True*, *maxdepth=None*, ***kwargs*)

Alias of *AbstractFileSystem.du*.

download(*rpath*, *lpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.get*.

du(*path*, *total=True*, *maxdepth=None*, *withdirs=False*, ***kwargs*)

Space used by files and optionally directories within a path

Directory size does not include the size of its contents.

Parameters

- **path** (*str*) –
- **total** (*bool*) – Whether to sum all the file sizes
- **maxdepth** (*int or None*) – Maximum number of directory levels to descend, None for unlimited.
- **withdirs** (*bool*) – Whether to include directory paths in the output.
- **kwargs** (passed to `find`) –

Returns

- **Dict of {path** (*size} if total=False, or int otherwise, where numbers*)
- *refer to bytes used.*

end_transaction()

Finish write transaction, non-context version

exists(*path*, ***kwargs*)

Is there a file at the given path

expand_path(*path*, *recursive=False*, *maxdepth=None*, ***kwargs*)

Turn one or more globs or directories into a list of all matching paths to files or directories.

kwargs are passed to `glob` or `find`, which may in turn call `ls`

find(*path*, *maxdepth=None*, *withdirs=False*, *detail=False*, ***kwargs*)

List all files below path.

Like posix `find` command without conditions

Parameters

- **path** (*str*) –
- **maxdepth** (*int or None*) – If not None, the maximum number of levels to descend
- **withdirs** (*bool*) – Whether to include directory paths in the output. This is True when used by glob, but users usually only want files.
- **ls.** (*kwargs are passed to*) –

static from_json(*blob*)

Recreate a filesystem instance from JSON representation

See `.to_json()` for the expected structure of the input

Parameters
blob (*str*) –

Return type
file system instance, not necessarily of this particular class.

property fsid

Persistent filesystem id that can be used to compare filesystems across sessions.

get(*rpath*, *lpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, *maxdepth=None*, ***kwargs*)

Copy file(s) to local.

Copies a specific file or tree of files (if recursive=True). If lpath ends with a "/", it will be assumed to be a directory, and target files will go within. Can submit a list of paths, which may be glob-patterns and will be expanded.

Calls get_file for each source.

get_file(*rpath*, *lpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, *outfile=None*, ***kwargs*)

Copy single remote file to local

get_mapper(*root=''*, *check=False*, *create=False*, *missing_exceptions=None*)

Create key/value store based on this file-system

Makes a MutableMapping interface to the FS at the given root path. See `fsspec.mapping.FSMap` for further details.

getxattr(*path*, *attr*)

Get user-defined metadata attribute

glob(*path*, *maxdepth=None*, ***kwargs*)

Find files by glob-matching.

If the path ends with '/', only folders are returned.

We support `"**"`, `"?"` and `"[..]"`. We do not support ^ for pattern negation.

The *maxdepth* option is applied on the first ** found in the path.

kwargs are passed to `ls`.

head(*path*, *size=1024*)

Get the first `size` bytes from file

info(*path*, ***kwargs*)

Give details of entry at path

Returns a single dictionary, with exactly the same information as `ls` would with `detail=True`.

The default implementation should calls ls and could be overridden by a shortcut. kwargs are passed on to `` `ls()``.

Some file systems might not be able to measure the file's size, in which case, the returned dict will include `'size': None`.

Returns

- **dict with keys** (*name (full path in the FS), size (in bytes), type (file,)*
- *directory, or something else) and other FS-specific keys.*

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.

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=True*, ***kwargs*)

Alias of *AbstractFileSystem.ls*.

ls(*path*, *detail=True*, ***kwargs*)

List objects at path.

This should include subdirectories and files at that location. The difference between a file and a directory must be clear when details are requested.

The specific keys, or perhaps a FileInfo class, or similar, is TBD, but must be consistent across implementations. Must include:

- full path to the entry (without protocol)
- size of the entry, in bytes. If the value cannot be determined, will be `None`.
- type of entry, "file", "directory" or other

Additional information may be present, appropriate to the file-system, e.g., generation, checksum, etc.

May use refresh=True|False to allow use of self._ls_from_cache to check for a saved listing and avoid calling the backend. This would be common where listing may be expensive.

Parameters

- **path** (*str*) –
- **detail** (*bool*) – if True, gives a list of dictionaries, where each is the same as the result of `info(path)`. If False, gives a list of paths (str).
- **kwargs** (*may have additional backend-specific options, such as version*) – information

Returns

- *List of strings if detail is False, or list of directory information*
- *dicts if detail is True.*

makedir(*path*, *create_parents=True*, ***kwargs*)

Alias of *AbstractFileSystem.mkdir*.

makedirs(*path*, *exist_ok=False*)

Recursively make directories

Creates directory at path and any intervening required directories. Raises exception if, for instance, the path already exists but is a file.

Parameters

- **path** (*str*) – leaf directory name
- **exist_ok** (*bool (False)*) – If False, will error if the target already exists

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.

mkdirs(*path*, *exist_ok=False*)

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=False*, *maxdepth=None*, ***kwargs*)

Move file(s) from one location to another

open(*path*, *mode='rb'*, *block_size=None*, *cache_options=None*, *compression=None*, ***kwargs*)

Return a file-like object from the filesystem

The resultant instance must function correctly in a context `with` block.

Parameters

- **path** (*str*) – Target file
- **mode** (*str like 'rb', 'w'*) – See builtin `open()`
- **block_size** (*int*) – Some indication of buffering - this is a value in bytes
- **cache_options** (*dict, optional*) – Extra arguments to pass through to the cache.
- **compression** (*string or None*) – If given, open file using compression codec. Can either be a compression name (a key in `fsspec.compression.compr`) or "infer" to guess the compression from the filename suffix.
- **encoding** (*passed on to TextIOWrapper for text mode*) –
- **errors** (*passed on to TextIOWrapper for text mode*) –
- **newline** (*passed on to TextIOWrapper for text mode*) –

pipe(*path*, *value=None*, ***kwargs*)

Put value into path

(counterpart to `cat`)

Parameters

- **path** (*string or dict(str, bytes)*) – If a string, a single remote location to put `value` bytes; if a dict, a mapping of {path: bytesvalue}.
- **value** (*bytes, optional*) – If using a single path, these are the bytes to put there. Ignored if `path` is a dict

pipe_file(*path*, *value*, ***kwargs*)

Set the bytes of given file

put(*lpath*, *rpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, *maxdepth=None*, ***kwargs*)

Copy file(s) from local.

Copies a specific file or tree of files (if recursive=True). If rpath ends with a "/", it will be assumed to be a directory, and target files will go within.

Calls put_file for each source.

put_file(*lpath*, *rpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, ***kwargs*)

Copy single file to remote

read_block(*fn*, *offset*, *length*, *delimiter=None*)

Read a block of bytes from

Starting at `offset` of the file, read `length` bytes. If `delimiter` is set then we ensure that the read starts and stops at delimiter boundaries that follow the locations `offset` and `offset + length`. If `offset` is zero then we start at zero. The bytestring returned WILL include the end delimiter string.

If offset+length is beyond the eof, reads to eof.

Parameters

- **fn** (*string*) – Path to filename
- **offset** (*int*) – Byte offset to start read
- **length** (*int*) – Number of bytes to read. If None, read to end.
- **delimiter** (*bytes (optional)*) – Ensure reading starts and stops at delimiter bytestring

Examples

```
>>> fs.read_block('data/file.csv', 0, 13)
b'Alice, 100\nBo'
>>> fs.read_block('data/file.csv', 0, 13, delimiter=b'\n')
b'Alice, 100\nBob, 200\n'
```

Use `length=None` to read to the end of the file. >>> fs.read_block('data/file.csv', 0, None, delimiter=b'n') # doctest: +SKIP b'Alice, 100nBob, 200nCharlie, 300'

See also:

`fsspec.utils.read_block()`

read_bytes(*path*, *start=None*, *end=None*, ***kwargs*)

Alias of *AbstractFileSystem.cat_file*.

read_text(*path*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Get the contents of the file as a string.

Parameters

- **path** (*str*) – URL of file on this filesystems
- **encoding** (same as *open*.) –
- **errors** (same as *open*.) –
- **newline** (same as *open*.) –

rename(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.mv*.

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.

rm_file(*path*)

Delete a file

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.

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

Return type
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 – The signed URL

Return type
str

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=1024*)

Get the last `size` bytes from file

to_json()

JSON representation of this filesystem instance

Returns
str – protocol (text name of this class's protocol, first one in case of multiple), args (positional args, usually empty), and all other kwargs as their own keys.

Return type
JSON structure with keys cls (the python location of this class),

touch(*path*, *truncate=True*, ***kwargs*)

Create empty file, or update timestamp

Parameters

- **path** (*str*) – file location
- **truncate** (*bool*) – If True, always set file size to 0; if False, update timestamp and leave file unchanged, if backend allows this

property transaction

A context within which files are committed together upon exit

Requires the file class to implement *.commit()* and *.discard()* for the normal and exception cases.

transaction_type

alias of `Transaction`

ukey(*path*)

Hash of file properties, to tell if it has changed

unstrip_protocol(*name: str*) → str

Format FS-specific path to generic, including protocol

upload(*lpath*, *rpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.put*.

url(*path*)

Get HTTP URL of the given path

walk(*path*, *maxdepth=None*, *topdown=True*, *on_error='omit'*, ***kwargs*)

Return all files belows path

List all files, recursing into subdirectories; output is iterator-style, like `os.walk()`. For a simple list of files, `find()` is available.

When topdown is True, the caller can modify the dirnames list in-place (perhaps using del or slice assignment), and walk() will only recurse into the subdirectories whose names remain in dirnames; this can be used to prune the search, impose a specific order of visiting, or even to inform walk() about directories the caller creates or renames before it resumes walk() again. Modifying dirnames when topdown is False has no effect. (see os.walk)

Note that the "files" outputted will include anything that is not a directory, such as links.

Parameters

- **path** (*str*) – Root to recurse into
- **maxdepth** (*int*) – Maximum recursion depth. None means limitless, but not recommended on link-based file-systems.
- **topdown** (*bool (True)*) – Whether to walk the directory tree from the top downwards or from the bottom upwards.
- **on_error** (*"omit", "raise", a collable*) – if omit (default), path with exception will simply be empty; If raise, an underlying exception will be raised; if callable, it will be called with a single OSError instance as argument
- **kwargs** (passed to `ls`) –

write_bytes(*path*, *value*, ***kwargs*)

Alias of *AbstractFileSystem.pipe_file*.

write_text(*path*, *value*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Write the text to the given file.

An existing file will be overwritten.

Parameters

- **path** (*str*) – URL of file on this filesystems
- **value** (*str*) – Text to write.

- **encoding** (same as *open*.) –
- **errors** (same as *open*.) –
- **newline** (same as *open*.) –

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*)

close()

Close file

Finalizes writes, discards cache

commit()

If not auto-committing, finalize file

discard()

Cancel in-progress multi-upload

Should only happen during discarding this write-mode file

fileno()

Return underlying file descriptor if one exists.

Raise OSError if the IO object does not use a file descriptor.

flush(*force=False*)

Write buffered data to backend store.

Writes the current buffer, if it is larger than the block-size, or if the file is being closed.

Parameters
force (*bool*) – When closing, write the last block even if it is smaller than blocks are allowed to be. Disallows further writing to this file.

info()

File information about this path

isatty()

Return whether this is an 'interactive' stream.

Return False if it can't be determined.

read(*length=-1*)

Return data from cache, or fetch pieces as necessary

Parameters
length (*int (-1)*) – Number of bytes to read; if <0, all remaining bytes.

readable()

Whether opened for reading

readinto(*b*)

mirrors builtin file's readinto method

https://docs.python.org/3/library/io.html#io.RawIOBase.readinto

readline()

Read until first occurrence of newline character

Note that, because of character encoding, this is not necessarily a true line ending.

readlines()

Return all data, split by the newline character

readuntil(*char=b'\n'*, *blocks=None*)

Return data between current position and first occurrence of char

char is included in the output, except if the end of the tile is encountered first.

Parameters

- **char** (*bytes*) – Thing to find
- **blocks** (*None or int*) – How much to read in each go. Defaults to file blocksize - which may mean a new read on every call.

seek(*loc*, *whence=0*)

Set current file location

Parameters

- **loc** (*int*) – byte location
- **whence** (*{0, 1, 2}*) – from start of file, current location or end of file, resp.

seekable()

Whether is seekable (only in read mode)

tell()

Current file location

truncate(*size=None*, */*)

Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by tell(). Return the new size.

url()

HTTP link to this file's data

writable()

Whether opened for writing

write(*data*)

Write data to buffer.

Buffer only sent on flush() or if buffer is greater than or equal to blocksize.

Parameters
data (*bytes*) – Set of bytes to be written.

writelines(*lines*, */*)

Write a list of lines to stream.

Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

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 2023.6.0

- allow raw/session token for auth (#554)
- fix listings_expiry_time kwargs (#551)
- allow setting fixed metadata on put/pipe (#550)

7.4.2 2023.5.0

- Allow emulator host without protocol (#548)
- Prevent upload retry from closing the file being sent (#540)

7.4.3 2023.4.0

No changes

7.4.4 2023.3.0

- Don't let find() mess up dircache (#531)
- Drop py3.7 (#529)
- Update docs (#528)
- Make times UTC (#527)
- Use BytesIO for large bodies (#525)
- Fix: Don't append generation when it is absent (#523)
- get/put/cp consistency tests (#521)

7.4.5 2023.1.0

- Support create time (#516, 518)
- defer async session creation (#513, 514)
- support listing of file versions (#509)
- fix `sign` following versioned split protocol (#513)

7.4.6 2022.11.0

- implement object versioning (#504)

7.4.7 2022.10.0

- bump fsspec to 2022.10.0 (#503)

7.4.8 2022.8.1

- don't install prerelease aiohttp (#490)

7.4.9 2022.7.1

- Try cloud auth by default (#479)

7.4.10 2022.5.0

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

7.4.11 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)

7.4.12 2022.02.0

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

7.4.13 2022.01.0

- update refname for versions (#442)

7.4.14 2021.11.1

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

7.4.15 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)

7.4.16 2021.10.1

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

7.4.17 2021.10.0

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

7.4.18 2021.09.0

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

7.4.19 2021.08.1

- don't retry 404s (#406)

7.4.20 2021.07.0

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

7.4.21 2021.06.1

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

7.4.22 2021.06.0

- negative ranges in cat_file (#394)

7.4.23 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)

7.4.24 Version 2021.04.0

- switch to calver and fsspec pin

7.4.25 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

7.4.26 Version 0.7.0

- async operations via aiohttp

7.4.27 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.

7.4.28 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).

7.4.29 Version 0.5.2

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

7.4.30 Version 0.5.1

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

7.4.31 Version 0.5.0

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

7.4.32 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

R

S

T

U

W