# S3Utils Documentation

***Release 0.6.1***

**Sep Ehr (Seperman)**

**Sep 27, 2017**

# Contents

**S3utils is a simple interface for dealing with Amazon S3**

CHAPTER 1

# Installation

Install from PyPi:

```
pip install s3utils
```

Python 2.7, 3.3, 3.4, 3.5 are supported.

CHAPTER 2

# Setup

## Normal Setup

```
>>> from s3utils import S3utils
>>> s3utils = S3utils(
... AWS_ACCESS_KEY_ID = 'your access key',
... AWS_SECRET_ACCESS_KEY = 'your secret key',
... AWS_STORAGE_BUCKET_NAME = 'your bucket name',
... S3UTILS_DEBUG_LEVEL = 1,  #change it to 0 for less verbose
... )
```

or if you are using Django, simply:

## Django Setup

```
>>> S3UTILS_DEBUG_LEVEL=1
>>> AWS_ACCESS_KEY_ID = 'your access key'
>>> AWS_SECRET_ACCESS_KEY = 'your secret key'
>>> AWS_STORAGE_BUCKET_NAME = 'your bucket name'
```

**And in your code:**

```
>>> from s3utils import S3utils
>>> s3utils = S3utils()
```

**If you want to overwrite your bucket name in your code from what it is in the Django settings:**

```
>>> from s3utils import S3utils
>>> s3utils = S3utils(AWS_STORAGE_BUCKET_NAME='some other bucket')
```

CHAPTER 3

# Commands

The commands are made to be similar to Linux file commands:

cp, mv, chmod, ls, ll, echo, mkdir, rm

There are some Cloudfront specific commands too:

invalidate

# CHAPTER 4

## S3utils 0.6.1

**class** s3utils.**S3utils**(*AWS_ACCESS_KEY_ID=''*, *AWS_SECRET_ACCESS_KEY=''*, *AWS_STORAGE_BUCKET_NAME=''*, *S3UTILS_DEBUG_LEVEL=0*, *AWS_HEADERS={}*)

S3 Utils

A simple user friendly interface to Amazon S3. S3 utils methods are made similar to Linux commands so it is easier to use/remember for simple file operations on S3 buckets.

**chmod**(**args*, ***kwargs*)

sets permissions for a file on S3

**target_file** [string] Path to file on S3

**acl** [string, optional] File permissions on S3. Default is public-read

**options:**

- private: Owner gets FULL_CONTROL. No one else has any access rights.
- public-read: Owners gets FULL_CONTROL and the anonymous principal is granted READ access.
- public-read-write: Owner gets FULL_CONTROL and the anonymous principal is granted READ and WRITE access.
- authenticated-read: Owner gets FULL_CONTROL and any principal authenticated as a registered Amazon S3 user is granted READ access

```
>>> s3utils.chmod("path/to/file","private")
```

**connect**()

Establish the connection. This is done automatically for you.

If you lose the connection, you can manually run this to be re-connected.

**connect_cloudfront**()

Connect to Cloud Front. This is done automatically for you when needed.

**cp**(*local_path*, *target_path*, *acl='public-read'*, *del_after_upload=False*, *overwrite=True*, *invalidate=False*)

Copy a file or folder from local to s3.

**local_path** [string] Path to file or folder. Or if you want to copy only the contents of folder, add /* at the end of folder name

**target_path** [string] Target path on S3 bucket.

**acl** [string, optional] File permissions on S3. Default is public-read

**options:**

- private: Owner gets FULL_CONTROL. No one else has any access rights.
- public-read: Owners gets FULL_CONTROL and the anonymous principal is granted READ access.
- public-read-write: Owner gets FULL_CONTROL and the anonymous principal is granted READ and WRITE access.
- authenticated-read: Owner gets FULL_CONTROL and any principal authenticated as a registered Amazon S3 user is granted READ access

**del_after_upload** [boolean, optional] delete the local file after uploading. This is effectively like moving the file. You can use s3utils.mv instead of s3utils.cp to move files from local to S3. It basically sets this flag to True. default = False

**overwrite** [boolean, optional] overwrites files on S3 if set to True. Default is True

**invalidate** [boolean, optional] invalidates the CDN (a.k.a Distribution) cache if the file already exists on S3 default = False Note that invalidation might take up to 15 minutes to take place. It is easier and faster to use cache buster to grab lastest version of your file on CDN than invalidation.

**Returns**

Nothing on success but it will return what went wrong if something fails.

```
>>> s3utils.cp("path/to/folder","/test/")
copying /path/to/myfolder/test2.txt to test/myfolder/test2.txt
copying /path/to/myfolder/test.txt to test/myfolder/test.txt
copying /path/to/myfolder/hoho/photo.JPG to test/myfolder/hoho/photo.JPG
copying /path/to/myfolder/hoho/haha/ff to test/myfolder/hoho/haha/ff
```

```
>>> # When overwrite is set to False, it returns the file(s) that were␣
↪already existing on s3 and were not overwritten.
>>> s3utils.cp("/tmp/test3.txt", "test3.txt", overwrite=False)
ERROR:root:test3.txt already exist. Not overwriting.
>>> {'existing_files': {'test3.txt'}}
```

```
>>> # To overwrite the files on S3 and invalidate the CDN (cloudfront) cache␣
↪so the new file goes on CDN:
>>> s3utils.cp("path/to/folder","/test/", invalidate=True)
copying /path/to/myfolder/test2.txt to test/myfolder/test2.txt
copying /path/to/myfolder/test.txt to test/myfolder/test.txt
copying /path/to/myfolder/hoho/photo.JPG to test/myfolder/hoho/photo.JPG
copying /path/to/myfolder/hoho/haha/ff to test/myfolder/hoho/haha/ff
```

```
>>> # When file does not exist, it returns a dictionary of what went wrong.
>>> s3utils.cp("/tmp/does_not_exist", "somewhere")
ERROR:root:trying to upload to s3 but file doesn't exist: /tmp/does_not_exist
>>> {'file_does_not_exist': '/tmp/does_not_exist'}
```

**cp_cropduster_image**(**args*, ***kwargs*)

Deal with saving cropduster images to S3. Cropduster is a Django library for resizing editorial images. S3utils was originally written to put cropduster images on S3 bucket.

**MEDIA_ROOT** [string] Django media root. Currently it is ONLY used in cp_cropduster_image method. NOT any other method as this library was originally made to put Django cropduster images on s3 bucket.

**S3_ROOT_BASE** [string] S3 media root base. This will be the root folder in S3. Currently it is ONLY used in cp_cropduster_image method. NOT any other method as this library was originally made to put Django cropduster images on s3 bucket.

**disconnect**()

Close the connection.

This is normally done automatically when the garbage collector is deleting s3utils object.

**echo**(*content*, *target_path*, *acl='public-read'*, *overwrite=True*, *invalidate=False*)

Similar to Linux Echo command.

Puts the string into the target path on s3

**content** [string] The content to be put on the s3 bucket.

**target_path** [string] Target path on S3 bucket.

**acl** [string, optional] File permissions on S3. Default is public-read

**options:**

- private: Owner gets FULL_CONTROL. No one else has any access rights.
- public-read: (Default) Owners gets FULL_CONTROL and the anonymous principal is granted READ access.
- public-read-write: Owner gets FULL_CONTROL and the anonymous principal is granted READ and WRITE access.
- authenticated-read: Owner gets FULL_CONTROL and any principal authenticated as a registered Amazon S3 user is granted READ access

**overwrite** [boolean, optional] overwrites files on S3 if set to True. Default is True

**invalidate** [boolean, optional] invalidates the CDN (a.k.a Distribution) cache if the file already exists on S3 default = False Note that invalidation might take up to 15 minutes to take place. It is easier and faster to use cache buster to serve the lastest version of your file on CDN than invalidation.

**Returns:**

Nothing on success, otherwise it returns what went wrong.

Return type: dict

```
>>> # On success returns nothing:
>>> s3utils.echo("Hello World!","/test.txt")
>>> # On failure returns what went wrong
>>> s3utils.echo("Hello World!","/test/")
{'InvalidS3Path': "path on S3 can not end in /"}
```

**invalidate**(**args*, ***kwargs*)

Invalidate the CDN (distribution) cache for a certain file of files. This might take up to 15 minutes to be effective.

You can check for the invalidation status using check_invalidation_request.

```
>>> from s3utils import S3utils
>>> s3utils = S3utils(
... AWS_ACCESS_KEY_ID = 'your access key',
... AWS_SECRET_ACCESS_KEY = 'your secret key',
... AWS_STORAGE_BUCKET_NAME = 'your bucket name',
... S3UTILS_DEBUG_LEVEL = 1,  #change it to 0 for less verbose
... )
>>> aa = s3utils.invalidate("test/myfolder/hoho/photo.JPG")
>>> print(aa)
('your distro id', u'your request id')
>>> invalidation_request_id = aa[1]
>>> bb = s3utils.check_invalidation_request(*aa)
>>> for inval in bb:
...     print('Object: %s, ID: %s, Status: %s' % (inval, inval.id, inval.
↪status))
```

**ll** (*folder=''*, *begin_from_file=''*, *num=-1*, *all_grant_data=False*)
Get the list of files and permissions from S3.

This is similar to LL (ls -lah) in Linux: List of files with permissions.

**folder** [string] Path to file on S3

**num: integer, optional** number of results to return, by default it returns all results.

**begin_from_file** [string, optional] which file to start from on S3. This is usedful in case you are iterating over lists of files and you need to page the result by starting listing from a certain file and fetching certain num (number) of files.

**all_grant_data** [Boolean, optional] More detailed file permission data will be returned.

```
>>> from s3utils import S3utils
>>> s3utils = S3utils(
... AWS_ACCESS_KEY_ID = 'your access key',
... AWS_SECRET_ACCESS_KEY = 'your secret key',
... AWS_STORAGE_BUCKET_NAME = 'your bucket name',
... S3UTILS_DEBUG_LEVEL = 1,  #change it to 0 for less verbose
... )
>>> import json
>>> # We use json.dumps to print the results more readable:
>>> my_folder_stuff = s3utils.ll("/test/")
>>> print(json.dumps(my_folder_stuff, indent=2))
{
  "test/myfolder/": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    }
  ],
  "test/myfolder/em/": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    }
  ],
  "test/myfolder/hoho/": [
    {
      "name": "owner's name",
```

```
      "permission": "FULL_CONTROL"
    }
  ],
  "test/myfolder/hoho/.DS_Store": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    },
    {
      "name": null,
      "permission": "READ"
    }
  ],
  "test/myfolder/hoho/haha/": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    }
  ],
  "test/myfolder/hoho/haha/ff": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    },
    {
      "name": null,
      "permission": "READ"
    }
  ],
  "test/myfolder/hoho/photo.JPG": [
    {
      "name": "owner's name",
      "permission": "FULL_CONTROL"
    },
    {
      "name": null,
      "permission": "READ"
    }
  ],
}
```

**ls**(**args*, ***kwargs*)

gets the list of file names (keys) in a s3 folder

**folder** [string] Path to file on S3

**num: integer, optional** number of results to return, by default it returns all results.

**begin_from_file: string, optional** which file to start from on S3. This is usedful in case you are iterating over lists of files and you need to page the result by starting listing from a certain file and fetching certain num (number) of files.

```
>>> from s3utils import S3utils
>>> s3utils = S3utils(
... AWS_ACCESS_KEY_ID = 'your access key',
... AWS_SECRET_ACCESS_KEY = 'your secret key',
... AWS_STORAGE_BUCKET_NAME = 'your bucket name',
... S3UTILS_DEBUG_LEVEL = 1,  #change it to 0 for less verbose
... )
```

```
>>> print(s3utils.ls("test/"))
{u'test/myfolder/', u'test/myfolder/em/', u'test/myfolder/hoho/', u'test/
↪myfolder/hoho/.DS_Store', u'test/myfolder/hoho/haha/', u'test/myfolder/hoho/
↪haha/ff', u'test/myfolder/hoho/haha/photo.JPG'}
```

**mkdir** (**args*, ***kwargs*)
Create a folder on S3.

```
>>> s3utils.mkdir("path/to/my_folder")
Making directory: path/to/my_folder
```

**mv** (*local_file*, *target_file*, *acl='public-read'*, *overwrite=True*, *invalidate=False*)
Similar to Linux mv command.

Move the file to the S3 and deletes the local copy

It is basically s3utils.cp that has del_after_upload=True

```
>>> s3utils.mv("path/to/folder","/test/")
moving /path/to/myfolder/test2.txt to test/myfolder/test2.txt
moving /path/to/myfolder/test.txt to test/myfolder/test.txt
moving /path/to/myfolder/hoho/photo.JPG to test/myfolder/hoho/photo.JPG
moving /path/to/myfolder/hoho/haha/ff to test/myfolder/hoho/haha/ff
```

**Returns:**

Nothing on success, otherwise what went wrong.

Return type: dict

**rm** (**args*, ***kwargs*)
Delete the path and anything under the path.

```
>>> s3utils.rm("path/to/file_or_folder")
```

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

CHAPTER 6

# Author

Seperman (Sep Ehr)

- Zepworks
- Github
- Linkedin

# Python Module Index

## S

# Index