CFS_Manager Documentation

Release 1.0.0

Alison Streete

Dec 20, 2017

Contents

Contents

CHAPTER 1

Read Me

CFS_Manager aims to make cloud storage easy to manage, even when you're dealing with multiple providers. It creates a unified API for upload/download/modification of files on multiple platforms. The unified API is mostly internal to the script, as the main user interface is the console script (cli.py).

The most recent release supports simultaneous integration with up to four storage providers: Google, Dropbox, Box, and pCloud. In addition, there is a clear and documented process for adding support for new providers, so a rapid expansion in supported platforms should be expected. Once setup, you should be able to treat all your storage providers as if they were a unified whole, with more space than any one would have individually.

1.1 Setup And Installation

To use CFS_Manager, you need to have Python 3 and pip installed. Then run: `pip install cfs_manager`. The installation and dependencies will then be handled automatically. (Note: As CFS_Manager interacts with the SDKs of multiple cloud storage providers, these will also be installed as dependencies.)

Warning: The Dropbox SDK claims to be incompatible with Python <3.4. While problems haven't been reproduced, managing Dropbox using CFS_Manager is not recommended unless you have Python 3.4 or higher.

After installation, you'll have new system-level commands you can use to control CFS_Manager. To enable the CLI to interact with your cloud accounts, run `cfs-config` in the shell to set up your system. The console dialogue should walk you through providing settings. If you ever want to change your settings, just run `cfs-config` again. After that, running `cfs-manager` or `cfsm` will allow you to interact with the file manager directly.

If you want to add a folder to your list of managed directories, you can just drag cfs_watcher.py into that folder and run it. Alternatively, you can navigate to a directory and then run `cfs-watch`. If you do either of these, it will also create a zipper.ignore file in the directory. This is a regular text file you can use to list any files or directories that shouldn't be uploaded. As a last resort, you can manually add the file path to a 'managed.txt' file in CFS_Manager's installed directory.

This utility aims for OS-independence and should work on Windows, Mac, and Linux (at least). (In case you encounter OS-specific issues, please note your OS in bug reports so attempts to reproduce it go smoothly.)

1.2 About

CFS_Manager uses the Apache 2.0 license and is publicly hosted on github. It is developed by Alison Streete, but welcomes anyone interested in contributing. (You could see your name on this line!)

Questions? Bugs? Kudos? Confusion? Want to buy the developer a drink? Email her at alison.streete {@} gmail.com. *(If it's a bug, pull requests work too.)*

See the About CFS_Manager page for more details.

CHAPTER 2

About CFS_Manager

2.1 Purpose

CFS_Manager makes dealing with multiple file systems easy by letting you pretend you're only dealing with one of them. The idea for a cloud-multiplexing application grew out of a desire to backup files without having to buy a cloud storage subscription.

It was clear that there was more than enough disk space out there, in the form of free storage providers, for most people to back up all their files. The problem was that you'd have to get a little storage each from many providers, and manage each account seperately, which is a lot of mental overhead.

But what if you could just merge all of those accounts into a single, unified storage area? What if you didn't need to remember to check 20 different accounts individually and remember which file was in which container? What if that was all handled automatically?

Thus, CFS_Manager was born. As projects go, it's clearly more work to develop an application than to just track the files in each cloud directly. However, if anyone else has a problem like this one, CFS_Manager will make life far easier for *them* - even if not for its developers.

2.2 Getting To Initial Release

All functionality present in 1.0 was written in the seven days following Thanksgiving, and then tested and documented for two days after. From the developer's perspective, if something works, it should be publicly available for others to benefit from (and critique) it. Development should obviously continue beyond initial release, but when the point of stable usability is achieved, everyone should have access.

However, this means a lot of future functionality is still on the table at this point (there's an existing to-do list), and you're encouraged to suggest more. Furthermore, there are probably bugs left to squish, and lots of performance enhancements to make. If you'd like to help make CFS_Manager better, please read the Contributor Notes.

2.3 About The Developer

Alison Streete is a novice developer originally from the Caribbean.

She moved to Palo Alto, California this year and quickly assimilated to the local customs, such as software engineering. She's been teaching herself programming and computer science for the past three months. This is the first open, full-featured project she's created.

Please let her know if there's anything she can improve - Both in code quality and general project administration. She tries to follow best practices where she knows them and will check out any style guides you can think of.

2.4 About The Docs

The documentation for v1.0 was entirely written by Alison, mostly on Dec 3rd. As such, it's not as comprehensive as would be ideal. Ideally, no one who uses CFS_Manager should be confused for longer than it takes do a search. As such, anyone who uses this utility - at any level of technical proficiency - is strongly encouraged to contribute to the documentation.

If there's anything you thing should be rephrased, suggest the rephrase! If anything confused you, complain about it! If there's something that you thought wasn't intuitive enough, make a note of it! Obviously, CFS_Manager tries to be clear and easy to use. However, for any case where someone could possibly be confused, they shouldn't *stay* confused long.

2.4.1 On Spelling And Punctuation

Throughout the console interfaces, an attempt was made to use American spelling conventions, as they're more typical of the industry. However, I [the author] naturally write in (approximately) British English, and writing out the entire docs in faux-American would have been tiring. As such, you should expect the docs to predominantly use British spelling and punctuation conventions.

However, if you would like to contribute documentation, feel free to use either British or American conventions. A haphazard mix of inconsistent spelling choices will just make the project seem Canadian. /s

---

CHAPTER 3

Features List

This page is for cataloguing all the exciting features of CFS_Manager.

Planned future features are discussed here

3.1 System Commands

After installing CFS_Manager, you should be able to run any of these in your shell/terminal/command-prompt

- `cfs-config`

Starts the configuration flow for CFS_Manager, allowing the user to choose defaults and allow access to their preferred storage providers.

- `cfsm` or `cfs-manager`

Starts the CFS_Manager command line interface in the terminal window that is currently open.

- `cfs-do [command]`

Initialises the virtual file system, performs the specified command, and then terminates. Allows the execution of one-off commands (such as downloading a specific file) without entering the embedded command line interface.

- `cfs-watch`

Adds the current directory to the list of managed directories. Thus, its contents will be included any time a command to upload all managed files is given.

3.2 User Interface

For the specific commands behind each of these options, open the CFS_Manager command line and use '–commands' for a full list

- Upload and download files and folders.

- List all the files in your virtual file system.
- Display the metadata associated with any given file, such as which storage provider it's hosted by and the date of last modification.
- Optional autocompletion of filenames, to avoid having to write out really long titles.
- Delete individual files or wipe all files from your virtual file system (with confirmation dialogue).
- Display the amount of cloud space currently occupied by CFS_Manager, and the amount of free space available across platforms.
- Detailed help page, about page, and list of CLI commands and flags.
- Launch the project's license, documentation, or github repository from the command line.

CHAPTER 4

F.A.Q

4.1 Getting Started

How do I get CFS_Manager?

The process for installing CFS_Manager with pip is explained in the README. After that, just open your terminal/shell/command prompt and run `cfs-config`. The configuration flow will walk you through the rest.

If you don't yet have Python and pip, you can get the most recent version of Python here. You should get whatever the most recent 3.x.x release is. Once you have that, pip should come pre-installed, and you'll be good to go.

When I run 'cfsm' I get an error saying there's no 'system_config.txt'. What do I do?

This means that your cloud file system hasn't been configured yet. This is important because, without configuration, how can CFS_Manager know what cloud storage accounts to connect to? To fix this, you just need to run `cfs-config` the same way you'd run `cfsm`. Then the configuration dialogue should walk you through exactly what you need to do.

Do I need to run cfs-config again after upgrading versions?

If you upgrade from one version of CFS_Manager to another, you shouldn't have to re-do configuration. All of your settings are saved during the upgrade process, so you should be able to jump right back into your workflow without a second thought!

The only times you might need to run `cfs-config` again are if you want *different* settings (eg: a new default download location) or if CFS_Manager has added support for a new storage provider and you want to hook up that account.

What's the difference between 'cfsm' and 'cfs-do?'

The `cfsm` shell command launches the CFS_Manager command line. It transfers you from your operating system's console interface to one that only talks to your cloud management system. After running it, connections to your cloud accounts will be initialised and you'll be able to issue as many commands as you want, line by line.

`cfs-do` also let's you talk to the cloud manager, but it doesn't launch the command line dialogue. Instead, it lets you issue individual, one-off commands without breaking your workflow. You can just write `cfs-do --download example_doc.txt` to download something without having to first enter the CLI to give the '–download' command.

4.2 File Management

How do managed directories work?

The managed directories are all the folders that CFS_Manager has been explicitly told about. If you use '–upload-all', the application will upload all your files from each of those folders.

If you want to add a folder to this list, there are multiple options. One is to navigate to the relevant directory in your shell and then run the `cfs-watch` command. A more graphical way to do this would be to drag-and-drop a copy of the cfs-watcher.py file from the cfs_manager directory into the one you want to manage, and then run that file. Finally, you can directly edit the managed.txt file in the cfs_manager directory to add a new folder to the list.

What is a zipper.ignore file?

If you use an automatic method to add a managed directory, this will create a 'zipper.ignore' file. What this does is list all the files and directories that cfs_manager shouldn't upload. This tells the 'zipper' module, which compresses all the files for upload, that it should ignore everything in that list. 'zipper.ignore' is a regular text file, so you can add things to it if you want.

If you have a 'zipper.ignore' file in any directory, then cfs_manager will ignore the files it lists, regardless of what upload method you use. Whether you use '–upload-all' or '–upload [directory name]'; whether the folder is in your managed.txt or not. You can create one yourself and put it anywhere and it'll work the same.

Can I drag files into the CFS_Manager folder to manage them?

As of 1.2.x, this functionality is unsupported. Please check back here in case we get a chance to add it.

4.3 Interface

How does file name autocompletion work?

To autocomplete a file name, you need to write out the command you want to use and the beginning of the file name. After the partial file name, you should add the '-c' modifier. Make sure there's a space between the partial filename and '-c'!

For example, if you have the file 'my fly mixtape track 1.mp3', you can download it by typing `--download 'my fly' -c`. However, this only works if there's only one thing that could possibly match! If you also have the file 'my flying leason.mov' then, because they both begin with 'my fly', they might be confused with each other and you won't know which one will get downloaded. To avoid this, please write out enough of the name that no other file will match with it.

Can I use autocomplete with uploads?

Unfortunately, no. CFS_Manager knows the name of every file in your cloud system, but it does *not* know the name of every file on your hard drive. Getting a list of all of those would take a *very* long time. Since you can upload a file from any location on your computer, the app can't possibly guess which one you might mean, so you'll have to write the whole thing out.

How does the 'verbose' command work?

If you use the '–help' command, either on its own (to get the system-level help) or after a command (to get the help info for that command), you'll get the a reasonably short response. The default help statement for each command is just one line, and the default help file is half a dozen.

But what happens if that isn't enough information? You know what a command does, but not exactly what information you have to give for that command to do it's job? ("Do I give it a file name? A file path? How many arguments do I feed it?...") Or what if you want the more detailed explain-like-I'm-five help file? In that case, you should follow the '–help' command with the '–verbose' flag. Then you'll get the much wordier version. (If that *still* isn't enough, you can always use '–docs' to get to the documentation.)

Also, if you use '–verbose' after '–commands', you'll get the long-form description of *every* command. Which will probably take up your screen, but at least you'll have *all* the info.

4.4 Development

What versioning system does the project use?

CFS_Manager follows Semantic Versioning. That means that the version number comes in three parts - Major.Minor.Patch - with each number indicating a different degree of change.

- The patch number is changed when new bug fixes, performance improvements, and other background-level contributions are made.
- The minor version is incremented when new functionality is added that doesn't interfere with existing functionality.
- The major version is incremented when the interface is changed in ways that make things that used to work stop working.

Notably, the interface here refers to the command line interface. Anything that you can do in the CLI today should work just the same in the future, if the major version is the same. However, there is also a plugin architecture in the works which will likely be less stable, so there may be breaking changes to the plugin system in minor version - at least until plugins are out of alpha.

What should I do if I want to contribute?

At the moment, your best bets are either creating an issue on GitHub or emailing Alison directly. Then you can share your ideas, suggest new features, and figure out what sub-problems to work on.

If you want to learn more about the general guidelines around contributions, please read the Contributor Notes. If you want a problem to tackle *right now*, please check out the Project Direction page. It has a list of open problems and goals for the project, and if you tackle one of them the project will be forever grateful.

Is there a standard release schedule?

Nope! New releases are made when new features, bug fixes, and general improvements have been made and vetted.

CHAPTER 5

Contributor Notes

5.1 Project General

This project follows Semantic Versioning. As it is currently in 1.x, you can be confident that console commands that work now will keep working in other 1.x versions. As the internal API is not currently intended for out-of-app use, there may be breaking changes in minor versions. However, the internal API should remain stable through patch versions, and future releases that expose this API should fully stabilise it.

5.2 Internal Design Policy

CFS_Manager follows an internal interaction model where commands from the CLI call upon the top-level manager, which calls upon file-system-object abstractions of each storage provider. These file system objects then interface with the provider's API via a wrapper written to increase the interface similarity, from the internal perspective, between the different providers. Generally, the provider's official Python SDK sits between the in-house wrapper and the provider's API.

It is designed to be easily extended with new service providers, and ensures this by providing a very specific interface for the implementation of new file system classes. All file systems inherit from the carefully described Cloud File System class, and their methods are called interchangably by the top-level manager.

It anticipates lower variance as information cascades upward. Each wrapper is extremely ideosyncratic because it talks to an API with a unique design philosophy. However, they use similar functions to pass similar types of data to their file system object. The file system object manipulate this data with different internal logic, but passing the same results up to the manager. The manager accepts data in several formats, but collapses them all into a standardised format. And when the CLI displays data to the user or accepts their commands, that user should see no indication that the provider APIs are all wildly different.

5.3 Comment Policy

The general approach taken is to err on the side of too much commentary. The goal is to make it easy to know exactly what's happening upon skimming a plain-English description. Hopefully, one should not have to read the entire codebase to know what they need to change to improve their workflow or make a pull request. If you benefit from the current level of descriptiveness, please pass it on by including comments in your commits!

Direct contributions to the documentation or the UI text are considered as valuable as code contribution. If there was anything you had to figure out on your own that you wish had been in the standard docs, please write it up! The world will thank you (or, at least, Alison will).

5.4 Adding Support For Storage Providers

While all current support is built on top of externally sourced wrappers, this is not a requirement for contribution. If you'd like to build integration for the bare REST API into this project, please go ahead! In fact, the use of a more minimal wrapper that only supports the operations CFS_Manager requires would make the whole project lighter-weight.

When adding an externally-sourced SDK to the dependency tree, check the license! It has to be compatible with imports from an Apache 2.0 project. Preferred licenses for linked SDKs include the Apache, MIT, BSD, and ISC. (The LGPL is also compatible, but full GPL is not.)

If you'd like to suggest another storage provider to integrate, but don't want to write the code, go ahead. However, if Alison is the one writing the code, she'll prioritise providers based on how simple the integration seems. Providers with up-to-date SDKs and readable documentation go to the top of the stack. If you're a fan/user/developer of a provider and really want support for it, and you want to speed it up, the best way is to write a wrapper yourself. (The second best way is to improve their documentation and be on call for CFS_Manager developers to question you.)

5.4.1 Storage Provider Integration Process

If you add a new provider, there are a few places where updates will need to happen to fully integrate them:

1. You'll need to write the in-project wrapper. The current style of the project involves writing an internal wrapper to talk to external ones. Any new wrappers written should losely comply with the conventions of the other ones in terms of function names and what results they achieve.
2. You'll need to write a new class in file_systems.py. This class must inherit from CloudFileSystem and include the same methods. What those methods do is stated in their doc strings in the CloudFileSystem class definition. The consistency is necessary, as it's part of the internal API. Those methods will be called, by name, by manager.py, which is a high-level abstraction over all the cloud accounts and must treat them interchangeably.
3. You'll need to import this class to manager.py, and add it to the list 'fs_classes'. This will be how the manager instantiates these filesystems.
4. Create a function to convert the file metadata returned by the specific API to the general metadata format used by the manager.
5. Depending on what the access credentials needed by the file system are, you should add a setup step to the configuration.py. Ideally, users should save as much reusable info as possible (passwords excepted). Configuration imports fs_classes from manager.py, so your addition will automatically show up in the list of file systems for users to approve. However, you'll still need a step to write its name to the system_configuration.txt file.

Once you've completed those five steps, the newly-added filesystem should work just as well as all the others.

CHAPTER 6

Project Direction

There are many areas of the CFS_Manager project that are in need of development. If you'd like to tackle one of these areas, go ahead and make a pull request. If you'd like to add to this list, you can open an issue on GitHub. You can also contact Alison directly with any feedback or suggestions.

General contribution guidelines can be found here.

6.1 Open Issues

6.1.1 *Bugs*

- Inability to store files of the same name
- Autocomplete occasionally returns names that are missing their first letter(s)

6.1.2 *Performance Drains*

- Cloud systems must be accessed sequentially
- Zipping files is slow. (According to cProfile, the slowest part of the package that isn't network-access.)

6.2 Goals

6.2.1 *Recurring*

- Writing/editing documentation
- Writing new unit tests *(please help!)*
- Integrating more storage providers

- Test on new operating systems

6.2.2 *Near-term Enhancements*

- Open cloud files on local machine
- Migrating to a database using SQLAlchemy
- Migrating to standard Python CLI libraries (eg, argparse)
- When multiple files share a name, return list for user confirmation
- Allow users to choose storage order
- Make files that are dropped into a CFS_Manager folder managable

6.2.3 *Medium-term Enhancements*

- Allow file encryption
- Create architecture for plugins
- Enable users to sign up for new storage providers
- Enable multi-user shared folder access

6.2.4 *Stretch Goals*

- Have network connections to each cloud run as concurrent processes
- Programmatically determine the best deal on buying extra storage
- Create a GUI for file management
- Make the cloud filesystem locally mountable

CHAPTER 7

License

7.1 Software Licensing

All modules and dependencies of this package are Free and Open Source Software. The licenses of the dependencies may vary and should be consulted separately. All of them are under permissive free licenses.

All .py files contained in this package are released under the Apache License 2.0. The Apache License 2.0 should be considered the primary license of this project and the reference version is included below.

The Apache License 2.0 and the CC BY-SA 4.0 are both one-way compatibile with the GPLv3. As such, ALL works contained within this package may be integrated into GPL projects. Furthermore, the Apache License is approved by the FSF, the OSI, and the Debian Project.

7.1.1 Apache License 2.0

7.2 Documentation Licensing

In addition, all documentation and text files contained in or associated with this package are released under both the GFDL 1.3 and the CC BY-SA 4.0 license. The GFDL can be found here, and the CC BY-SA 4.0 can be found here.

7.2.1 GFDL Notice

CHAPTER 8

Change Log

This is where all inter-version changes in the CFS_Manager project will be noted. All changes noted here will be in summary form. If you're reading this on *readthedocs*, you can do a full-text search to get more details. This project follows Semantic Versioning.

8.1 1.x Series

8.1.1 1.3.0 - *2017-12-15*

Bugfixes

- No longer crashes from attempts to download non-existent files.

Performance Improvements

- Added rudimentary support for caching downloaded files to speed up repeated downloads

New Functionality

- Support for calling one-off commands from the shell

UI Improvements

- Improved error messages for file downloads

8.1.2 1.2.0 - *2017-12-8*

New Functionality

- Added support for the (confusingly named) Box cloud storage system.

8.1.3 1.1.1 - *2017-12-7*

Bugfixes

- Google Drive stops making a bunch of incorrectly named folders that it can't find.

UI Improvements

- More useful error message when cfs-config hasn't been run.

Added

- A unit testing module.

8.1.4 1.1.0.1

Bugfixes

- Setuptools incorrectly packaged the files which were uploaded to PyPI, so they had to be re-uploaded immediately. This should really not be considered a distinct version from 1.1.0, and is only a thing because of an upload error.

8.1.5 1.1.0 - *2017-12-5*

Bugfixes

- The documentation correctly ships with the PyPI package.
- GDrive no longer breaks when CFS_Manager is installed from PyPI.
- Large Dropbox uploads no longer cause crashes.

New Functionality

- Added filename autocompletion option.
- Enabled the 'cfs-config' system command for the configuration flow.
- Enabled the 'cfs-watch' system command to add the current directory to the set of managed directories.

UI Improvements

- Added a list of command options to the CLI command listing.

8.1.6 1.0.1 - *2017-12-4*

Bugfixes

- CLI tools are functional even if there's no managed.txt yet.

8.1.7 1.0.0 - *2017-12-3*

(Nothing - initial release)

CHAPTER 9

Tools

- search