pydata-google-auth Documentation

Release 0.1.0

PyData Development Team

Aug 01, 2023

CONTENTS

The `pydata_google_auth` module provides a wrapper to authenticate to Google APIs, such as Google BigQuery.

Contents:

CHAPTER

ONE

INSTALLATION

You can install pydata-google-auth with `conda`, `pip`, or by installing from source.

1.1 Conda

```
$ conda install pydata-google-auth --channel conda-forge
```

This installs pydata-google-auth and all common dependencies, including `google-auth`.

1.2 Pip

To install the latest version of pydata-google-auth: from the

```
$ pip install pydata-google-auth -U
```

This installs pydata-google-auth and all common dependencies, including `google-auth`.

1.3 Install from Source

```
$ pip install git+https://github.com/pydata/pydata-google-auth.git
```

1.4 Dependencies

This module requires following additional dependencies:

- google-auth: authentication and authorization for Google's API
- google-auth-oauthlib: integration with oauthlib for end-user authentication

CHAPTER

TWO

INTRODUCTION

pydata-google-auth wraps the google-auth and google-auth-oauthlib libraries to make it easier to get and cache user credentials for accessing the Google APIs from locally-installed data tools and libraries.

Warning: To use this module, you will need a Google account and developer project. Follow the Using the BigQuery sandbox instructions to get started with big data on Google Cloud without a credit card.

See the Google Cloud Platform authentication guide for best practices on authentication in production server contexts.

2.1 User credentials

Use the *`pydata_google_auth.get_user_credentials()`* to get user credentials, authenticated to Google APIs.

By default, pydata-google-auth will listen for the credentials on a local webserver, which is used as the redirect page from Google's OAuth 2.0 flow. When you set `use_local_webserver=False`, pydata-google-auth will request that you copy a token from the *Sign in to BigQuery* page.

2.1.1 Sign in to BigQuery

You are seeing this page because you are attempting to access BigQuery via one of several possible methods, including:

- the `pydata-google-auth` library

OR a `pandas` library helper function such as:

- `pandas.DataFrame.to_gbq()`
- `pandas.read_gbq()`

from this or another machine. If this is not the case, close this tab.

Enter the following verification code in the CommandLine Interface (CLI) on the machine you want to log into. This is a credential **similar to your password** and should not be shared with others.

Hint: You can close this tab when you're done.

2.2 Default credentials

Data library and tool authors can use the `pydata_google_auth.default()` function to get Application Default Credentials and fallback to user credentials when no valid Application Default Credentials are found.

When wrapping the `pydata_google_auth.default()` method for use in your tool or library, please provide your own client ID and client secret. Enable the APIs your users will need in the project which owns the client ID and secrets. Note that some APIs, such as Cloud Vision, bill the *client* project. Verify that the API you are enabling bills the user's project not the client project.

CHAPTER

THREE

COMMAND-LINE REFERENCE

Run the pydata_google_auth CLI with python -m pydata_google_auth.

```
usage: python -m pydata_google_auth [-h] {login,print-token} ...

Manage credentials for Google APIs.

optional arguments:
  -h, --help            show this help message and exit

commands:
  {login,print-token}
    login               Login to Google and save user credentials as a JSON
                        file to use as Application Default Credentials.
    print-token         Load a credentials JSON file and print an access token.
```

3.1 Saving user credentials with login

```
usage: python -m pydata_google_auth login [-h] [--scopes SCOPES]
                                          [--client_id CLIENT_ID]
                                          [--client_secret CLIENT_SECRET]
                                          [--use_local_webserver]
                                          destination

positional arguments:
  destination           Path of where to save user credentials JSON file.

optional arguments:
  -h, --help            show this help message and exit
  --scopes SCOPES       Comma-separated list of scopes (permissions) to
                        request from Google. See: https://developers.google.co
                        m/identity/protocols/googlescopes for a list of
                        available scopes. Default:
                        https://www.googleapis.com/auth/cloud-platform
  --client_id CLIENT_ID
                        (Optional, but recommended) Client ID. Use this in
                        combination with the --client-secret argument to
                        authenticate with an application other than the
                        default (PyData Auth). This argument is required to
```

(continues on next page)

(continued from previous page)

```
                        use APIs the track billing and quotas via the
                        application (such as Cloud Vision), rather than
                        billing the user (such as BigQuery does).
  --client_secret CLIENT_SECRET
                        (Optional, but recommended) Client secret. Use this in
                        combination with the --client-id argument to
                        authenticate with an application other than the
                        default (PyData Auth). This argument is required to
                        use APIs the track billing and quotas via the
                        application (such as Cloud Vision), rather than
                        billing the user (such as BigQuery does).
  --use_local_webserver
                        Use a local webserver for the user authentication.
                        This starts a webserver on localhost, which allows the
                        browser to pass a token directly to the program.
```

Save credentials with Cloud Platform scope to `~/keys/google-credentials.json`.

```
python -m pydata_google_auth login ~/keys/google-credentials.json
```

3.2 Loading user credentials with `print-token`

Print an access token associate with the credentials at `~/keys/google-credentials.json`.

```
python -m pydata_google_auth print-token ~/keys/google-credentials.json
```

Use `curl` and the `credentials.json` user credentials file to download the contents of `gs://your-bucket/path/to/object.txt` with the Google Cloud Storage JSON REST API.

```
curl -X GET \
    -H "Authorization: Bearer $(python -m pydata_google_auth print-token credentials.
↪json)" \
    "https://storage.googleapis.com/storage/v1/b/your-bucket/o/path%%2Fto%%2Fobject.txt?
↪alt=media"
```

CHAPTER

FOUR

API REFERENCE

default(scopes[, client_id, client_secret, ...])	Get credentials and default project for accessing Google APIs.
get_user_credentials(scopes[, client_id, ...])	Gets user account credentials.
load_user_credentials(path)	Gets user account credentials from JSON file at `path`.
save_user_credentials(scopes, path[, ...])	Gets user account credentials and saves them to a JSON file at `path`.
load_service_account_credentials(path[, scopes])	Gets service account credentials from JSON file at `path`.
cache.CredentialsCache()	Shared base class for crentials classes.
cache.READ_WRITE	Write credentials to disk and read cached credentials from disk.
cache.REAUTH	Write credentials to disk.
cache.NOOP	Noop impmentation of credentials cache.
exceptions.PyDataCredentialsError	Raised when invalid credentials are provided, or tokens have expired.

`pydata_google_auth.`**`default`**(*scopes*, *client_id=None*, *client_secret=None*, *credentials_cache=<pydata_google_auth.cache.ReadWriteCredentialsCache object>*, *use_local_webserver=True*, *auth_local_webserver=None*, *redirect_uri=None*)

Get credentials and default project for accessing Google APIs.

This method first attempts to get credentials via the `google.auth.default()` function. If it is unable to get valid credentials, it then attempts to get user account credentials via the *`pydata_google_auth.get_user_credentials()`* function.

Parameters

- **`scopes`** (*`list[str]`*) – A list of scopes to use when authenticating to Google APIs. See the list of OAuth 2.0 scopes for Google APIs.
- **`client_id`** (*`str, optional`*) – The client secrets to use when prompting for user credentials. Defaults to a client ID associated with pydata-google-auth.

  If you are a tool or library author, you must override the default value with a client ID associated with your project. Per the Google APIs terms of service, you must not mask your API client's identity when using Google APIs.
- **`client_secret`** (*`str, optional`*) – The client secrets to use when prompting for user credentials. Defaults to a client secret associated with pydata-google-auth.

  If you are a tool or library author, you must override the default value with a client secret associated with your project. Per the Google APIs terms of service, you must not mask your

API client's identity when using Google APIs.

- `credentials_cache` (`pydata_google_auth.cache.CredentialsCache`, *optional*) – An object responsible for loading and saving user credentials.

  By default, pydata-google-auth reads and writes credentials in `$HOME/.config/pydata/pydata_google_credentials.json` or `$APPDATA/.config/pydata/pydata_google_credentials.json` on Windows.

- `use_local_webserver` (*bool, optional*) – Use a local webserver for the user authentication `google_auth_oauthlib.flow.InstalledAppFlow`. Binds a webserver to an open port on `localhost` between 8080 and 8089, inclusive, to receive authentication token. If not set, defaults to `False`, which requests a token via the console.

- `auth_local_webserver` (*deprecated*) – Use the `use_local_webserver` parameter instead.

- `redirect_uri` (*str, optional*) – Redirect URIs are endpoints to which the OAuth 2.0 server can send responses. They may be used in situations such as

  - an organization has an org specific authentication endpoint
  - an organization can not use an endpoint directly because of constraints on access to the internet (i.e. when running code on a remotely hosted device).

Returns

credentials, project_id – credentials : OAuth 2.0 credentials for accessing Google APIs

project_id : A default Google developer project ID, if one could be determined from the credentials. For example, this returns the project ID associated with a service account when using a service account key file. It returns None when using user-based credentials.

Return type tuple[google.auth.credentials.Credentials, str or None]

Raises ***pydata_google_auth.exceptions.PyDataCredentialsError*** – If unable to get valid credentials.

`pydata_google_auth.`**`get_user_credentials`**(*scopes*, *client_id=None*, *client_secret=None*, *credentials_cache=<pydata_google_auth.cache.ReadWriteCredentialsCache object>*, *use_local_webserver=True*, *auth_local_webserver=None*, *redirect_uri=None*)

Gets user account credentials.

This function authenticates using user credentials, either loading saved credentials from the cache or by going through the OAuth 2.0 flow.

The default read-write cache attempts to read credentials from a file on disk. If these credentials are not found or are invalid, it begins an OAuth 2.0 flow to get credentials. You'll open a browser window asking for you to authenticate to your Google account using the product name `PyData Google Auth`. The permissions it requests correspond to the scopes you've provided.

Additional information on the user credentials authentication mechanism can be found here.

Parameters

- `scopes` (*list[str]*) – A list of scopes to use when authenticating to Google APIs. See the list of OAuth 2.0 scopes for Google APIs.

- `client_id` (*str, optional*) – The client secrets to use when prompting for user credentials. Defaults to a client ID associated with pydata-google-auth.

If you are a tool or library author, you must override the default value with a client ID associated with your project. Per the Google APIs terms of service, you must not mask your API client's identity when using Google APIs.

- **client_secret** (*str, optional*) – The client secrets to use when prompting for user credentials. Defaults to a client secret associated with pydata-google-auth.

  If you are a tool or library author, you must override the default value with a client secret associated with your project. Per the Google APIs terms of service, you must not mask your API client's identity when using Google APIs.

- **credentials_cache** (*pydata_google_auth.cache.CredentialsCache, optional*) – An object responsible for loading and saving user credentials.

  By default, pydata-google-auth reads and writes credentials in `$HOME/.config/pydata/pydata_google_credentials.json` or `$APPDATA/.config/pydata/pydata_google_credentials.json` on Windows.

- **use_local_webserver** (*bool, optional*) – Use a local webserver for the user authentication `google_auth_oauthlib.flow.InstalledAppFlow`. Binds a webserver to an open port on `localhost` between 8080 and 8089, inclusive, to receive authentication token. If not set, defaults to `False`, which requests a token via the console.

- **auth_local_webserver** (*deprecated*) – Use the `use_local_webserver` parameter instead.

- **redirect_uri** (*str, optional*) – Redirect URIs are endpoints to which the OAuth 2.0 server can send responses. They may be used in situations such as
  - an organization has an org specific authentication endpoint
  - an organization can not use an endpoint directly because of constraints on access to the internet (i.e. when running code on a remotely hosted device).

Returns **credentials** – Credentials for the user, with the requested scopes.

Return type google.oauth2.credentials.Credentials

Raises ***pydata_google_auth.exceptions.PyDataCredentialsError*** – If unable to get valid user credentials.

pydata_google_auth.**load_service_account_credentials**(*path*, *scopes=None*)

Gets service account credentials from JSON file at `path`.

Parameters

- **path** (*str*) – Path to credentials JSON file.
- **scopes** (*list[str], optional*) – A list of scopes to use when authenticating to Google APIs. See the list of OAuth 2.0 scopes for Google APIs.

Returns

Return type google.oauth2.service_account.Credentials

Raises ***pydata_google_auth.exceptions.PyDataCredentialsError*** – If unable to load service credentials.

Examples

Load credentials and use them to construct a BigQuery client.

```
import pydata_google_auth
import google.cloud.bigquery

credentials = pydata_google_auth.load_service_account_credentials(
    "/home/username/keys/google-service-account-credentials.json",
)
client = google.cloud.bigquery.BigQueryClient(
    credentials=credentials,
    project=credentials.project_id
)
```

`pydata_google_auth.`**`load_user_credentials`**`(`*path*`)`

Gets user account credentials from JSON file at `path`.

Parameters **`path`** (*`str`*) – Path to credentials JSON file.

Returns

Return type google.auth.credentials.Credentials

Raises *`pydata_google_auth.exceptions.PyDataCredentialsError`* – If unable to load user credentials.

Examples

Load credentials and use them to construct a BigQuery client.

```
import pydata_google_auth
import google.cloud.bigquery

credentials = pydata_google_auth.load_user_credentials(
    "/home/username/keys/google-credentials.json",
)
client = google.cloud.bigquery.BigQueryClient(
    credentials=credentials,
    project="my-project-id"
)
```

`pydata_google_auth.`**`save_user_credentials`**`(`*scopes*, *path*, *client_id=None*, *client_secret=None*, *use_local_webserver=True*`)`

Gets user account credentials and saves them to a JSON file at `path`.

This function authenticates using user credentials by going through the OAuth 2.0 flow.

Parameters

- **`scopes`** (*`list[str]`*) – A list of scopes to use when authenticating to Google APIs. See the list of OAuth 2.0 scopes for Google APIs.
- **`path`** (*`str`*) – Path to save credentials JSON file.
- **`client_id`** (*`str, optional`*) – The client secrets to use when prompting for user credentials. Defaults to a client ID associated with pydata-google-auth.

If you are a tool or library author, you must override the default value with a client ID associated with your project. Per the Google APIs terms of service, you must not mask your API client's identity when using Google APIs.

- **client_secret** (*str, optional*) – The client secrets to use when prompting for user credentials. Defaults to a client secret associated with pydata-google-auth.

  If you are a tool or library author, you must override the default value with a client secret associated with your project. Per the Google APIs terms of service, you must not mask your API client's identity when using Google APIs.

- **use_local_webserver** (*bool, optional*) – Use a local webserver for the user authentication `google_auth_oauthlib.flow.InstalledAppFlow`. Binds a webserver to an open port on `localhost` between 8080 and 8089, inclusive, to receive authentication token. If not set, defaults to `False`, which requests a token via the console.

Returns

Return type None

Raises ***pydata_google_auth.exceptions.PyDataCredentialsError*** – If unable to get valid user credentials.

Examples

Get credentials for Google Cloud Platform and save them to `/home/username/keys/google-credentials.json`.

```
pydata_google_auth.save_user_credentials(
    ["https://www.googleapis.com/auth/cloud-platform"],
    "/home/username/keys/google-credentials.json",
    use_local_webserver=True,
)
```

Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to use these credentials with Google Application Default Credentials.

```
export GOOGLE_APPLICATION_CREDENTIALS='/home/username/keys/google-credentials.json'
```

Caching implementations for reading and writing user credentials.

class pydata_google_auth.cache.**CredentialsCache**

Bases: `object`

Shared base class for crentials classes.

This class also functions as a noop implementation of a credentials class.

load()

Load credentials from disk.

Does nothing in this base class.

Returns Returns user account credentials loaded from disk or `None` if no credentials could be found.

Return type google.oauth2.credentials.Credentials, optional

save(*credentials*)

Write credentials to disk.

Does nothing in this base class.

Parameters **credentials** (*google.oauth2.credentials.Credentials*) – User credentials to save to disk.

`pydata_google_auth.cache.`**`NOOP = <pydata_google_auth.cache.CredentialsCache object>`**

Noop impmentation of credentials cache.

This cache always reauthorizes and never save credentials to disk. Recommended for shared machines.

`pydata_google_auth.cache.`**`READ_WRITE = <pydata_google_auth.cache.ReadWriteCredentialsCache object>`**

Write credentials to disk and read cached credentials from disk.

`pydata_google_auth.cache.`**`REAUTH = <pydata_google_auth.cache.WriteOnlyCredentialsCache object>`**

Write credentials to disk. Never read cached credentials from disk.

Use this to reauthenticate and refresh the cached credentials.

`class pydata_google_auth.cache.`**`ReadWriteCredentialsCache`**(*dirname='pydata', filename='pydata_google_credentials.json'*)

Bases: *pydata_google_auth.cache.CredentialsCache*

A *CredentialsCache* which writes to disk and reads cached credentials from disk.

Parameters

- **dirname** (*str, optional*) – Name of directory to write credentials to. This directory is created within the `.config` subdirectory of the `HOME` (`APPDATA` on Windows) directory.
- **filename** (*str, optional*) – Name of the credentials file within the credentials directory.

`load`()

Load credentials from disk.

Returns Returns user account credentials loaded from disk or `None` if no credentials could be found.

Return type google.oauth2.credentials.Credentials, optional

`save`(*credentials*)

Write credentials to disk.

Parameters **credentials** (*google.oauth2.credentials.Credentials*) – User credentials to save to disk.

`class pydata_google_auth.cache.`**`WriteOnlyCredentialsCache`**(*dirname='pydata', filename='pydata_google_credentials.json'*)

Bases: *pydata_google_auth.cache.CredentialsCache*

A *CredentialsCache* which writes to disk, but doesn't read from disk.

Use this class to reauthorize against Google APIs and cache your credentials for later.

Parameters

- **dirname** (*str, optional*) – Name of directory to write credentials to. This directory is created within the `.config` subdirectory of the `HOME` (`APPDATA` on Windows) directory.
- **filename** (*str, optional*) – Name of the credentials file within the credentials directory.

`save`(*credentials*)

Write credentials to disk.

Parameters **credentials** (*google.oauth2.credentials.Credentials*) – User credentials to save to disk.

exception pydata_google_auth.exceptions.**PyDataConnectionError**

Bases: `RuntimeError`

Raised when unable to fetch credentials due to connection error.

exception pydata_google_auth.exceptions.**PyDataCredentialsError**

Bases: `ValueError`

Raised when invalid credentials are provided, or tokens have expired.

CHAPTER

FIVE

CONTRIBUTING TO PYDATA-GOOGLE-AUTH

Table of contents:

5.1 Where to start?

All contributions, bug reports, bug fixes, documentation improvements, enhancements and ideas are welcome.

If you are simply looking to start working with the *pydata-google-auth* codebase, navigate to the GitHub "issues" tab and start looking through interesting issues.

Or maybe through using *pydata-google-auth* you have an idea of your own or are looking for something in the documentation and thinking 'this can be improved'... you can do something about it!

Feel free to ask questions on the mailing list.

5.2 Bug reports and enhancement requests

Bug reports are an important part of making *pydata-google-auth* more stable. Having a complete bug report will allow others to reproduce the bug and provide insight into fixing it. Because many versions of *pydata-google-auth* are supported, knowing version information will also identify improvements made since previous versions. Trying the bug-producing code out on the *master* branch is often a worthwhile exercise to confirm the bug still exists. It is also worth searching existing bug reports and pull requests to see if the issue has already been reported and/or fixed.

Bug reports must:

1. Include a short, self-contained Python snippet reproducing the problem. You can format the code nicely by using GitHub Flavored Markdown

```
>>> import pydata_google_auth
>>> creds, proj = pydata_google_auth.default(...)
...
```

2. Include the full version string of *pydata-google-auth*.

```
>>> import pydata_google_auth
>>> pydata_google_auth.__version__
...
```

3. Explain why the current behavior is wrong/not desired and what you expect instead.

The issue will then show up to the *pydata-google-auth* community and be open to comments/ideas from others.

5.3 Working with the code

Now that you have an issue you want to fix, enhancement to add, or documentation to improve, you need to learn how to work with GitHub and the *pydata-google-auth* code base.

5.3.1 Version control, Git, and GitHub

To the new user, working with Git is one of the more daunting aspects of contributing to *pydata-google-auth*. It can very quickly become overwhelming, but sticking to the guidelines below will help keep the process straightforward and mostly trouble free. As always, if you are having difficulties please feel free to ask for help.

The code is hosted on GitHub. To contribute you will need to sign up for a free GitHub account. We use Git for version control to allow many people to work together on the project.

Some great resources for learning Git:

- the GitHub help pages.
- the NumPy's documentation.
- Matthew Brett's Pydagogue.

5.3.2 Getting started with Git

GitHub has instructions for installing git, setting up your SSH key, and configuring git. All these steps need to be completed before you can work seamlessly between your local repository and GitHub.

5.3.3 Forking

You will need your own fork to work on the code. Go to the pydata-google-auth project page and hit the `Fork` button. You will want to clone your fork to your machine:

```
git clone git@github.com:your-user-name/pydata-google-auth.git pydata-google-auth-
↪yourname
cd pydata-google-auth-yourname
git remote add upstream git://github.com/pydata/pydata-google-auth.git
```

This creates the directory *pydata-google-auth-yourname* and connects your repository to the upstream (main project) *pydata-google-auth* repository.

The testing suite will run automatically on CircleCI once your pull request is submitted. However, if you wish to run the test suite on a branch prior to submitting the pull request, then CircleCI needs to be hooked up to your GitHub repository. Instructions for doing so are here..

5.3.4 Creating a branch

You want your master branch to reflect only production-ready code, so create a feature branch for making your changes. For example:

```
git branch shiny-new-feature
git checkout shiny-new-feature
```

The above can be simplified to:

```
git checkout -b shiny-new-feature
```

This changes your working directory to the shiny-new-feature branch. Keep any changes in this branch specific to one bug or feature so it is clear what the branch brings to *pydata-google-auth*. You can have many shiny-new-features and switch in between them using the git checkout command.

To update this branch, you need to retrieve the changes from the master branch:

```
git fetch upstream
git rebase upstream/master
```

This will replay your commits on top of the latest pydata-google-auth git master. If this leads to merge conflicts, you must resolve these before submitting your pull request. If you have uncommitted changes, you will need to `stash` them prior to updating. This will effectively store your changes and they can be reapplied after updating.

5.3.5 Install in Development Mode

It's helpful to install pydata-google-auth in development mode so that you can use the library without reinstalling the package after every change.

Conda

Create a new conda environment and install the necessary dependencies

```
$ conda create -n my-env --channel conda-forge  \
      google-auth-oauthlib \
      google-api-python-client \
      google-auth-httplib2
$ source activate my-env
```

Install pydata-google-auth in development mode

```
$ python setup.py develop
```

Pip & virtualenv

Skip this section if you already followed the conda instructions.

Create a new virtual environment.

```
$ virtualenv env
$ source env/bin/activate
```

You can install pydata-google-auth and its dependencies in development mode via pip.

```
$ pip install -e .
```

5.4 Contributing to the code base

Code Base:

5.4.1 Code standards

Writing good code is not just about what you write. It is also about *how* you write it. During testing on CircleCI, several tools will be run to check your code for stylistic errors. Generating any warnings will cause the test to fail. Thus, good style is a requirement for submitting code to *pydata-google-auth*.

In addition, because a lot of people use our library, it is important that we do not make sudden changes to the code that could have the potential to break a lot of user code as a result, that is, we need it to be as *backwards compatible* as possible to avoid mass breakages.

Python (PEP8)

pydata-google-auth uses the PEP8 standard. There are several tools to ensure you abide by this standard. Here are *some* of the more common `PEP8` issues:

- we restrict line-length to 79 characters to promote readability
- passing arguments should have spaces after commas, e.g. `foo(arg1, arg2, kw1='bar')`

CircleCI will run the flake8 tool and the 'black' code formatting tool to report any stylistic errors in your code. Therefore, it is helpful before submitting code to run the check yourself on the diff:

```
black .
git diff master | flake8 --diff
```

Backwards Compatibility

Please try to maintain backward compatibility. If you think breakage is required, clearly state why as part of the pull request. Also, be careful when changing method signatures and add deprecation warnings where needed.

5.4.2 Test-driven development/code writing

pydata-google-auth is serious about testing and strongly encourages contributors to embrace test-driven development (TDD). This development process "relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test." So, before actually writing any code, you should write your tests. Often the test can be taken from the original GitHub issue. However, it is always worth considering additional use cases and writing corresponding tests.

Adding tests is one of the most common requests after code is pushed to *pydata-google-auth*. Therefore, it is worth getting in the habit of writing tests ahead of time so this is never an issue.

Like many packages, *pydata-google-auth* uses pytest.

Running the test suite

The tests can then be run directly inside your Git clone (without having to install *pydata-google-auth*) by typing:

```
pytest tests/unit
pytest tests/system.py
```

The tests suite is exhaustive and takes around 20 minutes to run. Often it is worth running only a subset of tests first around your changes before running the entire suite.

The easiest way to do this is with:

```
pytest tests/path/to/test.py -k regex_matching_test_name
```

Or with one of the following constructs:

```
pytest tests/[test-module].py
pytest tests/[test-module].py::[TestClass]
pytest tests/[test-module].py::[TestClass]::[test_method]
```

For more, see the pytest documentation.

Testing on multiple Python versions

pydata-google-auth uses nox to automate testing in multiple Python environments. First, install nox.

```
$ pip install --upgrade nox
```

To run tests in all versions of Python, run *nox* from the repository's root directory.

5.4.3 Documenting your code

Changes should be reflected in the release notes located in `doc/source/changelog.rst`. This file contains an ongoing change log. Add an entry to this file to document your fix, enhancement or (unavoidable) breaking change. Make sure to include the GitHub issue number when adding your entry (using `` GH#1234 `` where *1234* is the issue/pull request number).

If your code is an enhancement, it is most likely necessary to add usage examples to the existing documentation. Further, to let users know when this feature was added, the `versionadded` directive is used. The sphinx syntax for that is:

```
.. versionadded:: 0.1.3
```

This will put the text *New in version 0.1.3* wherever you put the sphinx directive. This should also be put in the docstring when adding a new function or method.

5.5 Contributing your changes to *pydata-google-auth*

5.5.1 Committing your code

Keep style fixes to a separate commit to make your pull request more readable.

Once you've made changes, you can see them by typing:

```
git status
```

If you have created a new file, it is not being tracked by git. Add it by typing:

```
git add path/to/file-to-be-added.py
```

Doing 'git status' again should give something like:

```
# On branch shiny-new-feature
#
#       modified:   /relative/path/to/file-you-added.py
#
```

Finally, commit your changes to your local repository with an explanatory message. *pydata-google-auth* uses a convention for commit message prefixes and layout. Here are some common prefixes along with general guidelines for when to use them:

- ENH: Enhancement, new functionality
- BUG: Bug fix
- DOC: Additions/updates to documentation
- TST: Additions/updates to tests
- BLD: Updates to the build process/scripts
- PERF: Performance improvement
- CLN: Code cleanup

The following defines how a commit message should be structured. Please reference the relevant GitHub issues in your commit message using GH1234 or #1234. Either style is fine, but the former is generally preferred:

- a subject line with < *80* chars.
- One blank line.
- Optionally, a commit message body.

Now you can commit your changes in your local repository:

```
git commit -m
```

5.5.2 Combining commits

If you have multiple commits, you may want to combine them into one commit, often referred to as "squashing" or "rebasing". This is a common request by package maintainers when submitting a pull request as it maintains a more compact commit history. To rebase your commits:

```
git rebase -i HEAD~#
```

Where # is the number of commits you want to combine. Then you can pick the relevant commit message and discard others.

To squash to the master branch do:

```
git rebase -i master
```

Use the `s` option on a commit to `squash`, meaning to keep the commit messages, or `f` to `fixup`, meaning to merge the commit messages.

Then you will need to push the branch (see below) forcefully to replace the current commits with the new ones:

```
git push origin shiny-new-feature -f
```

5.5.3 Pushing your changes

When you want your changes to appear publicly on your GitHub page, push your forked feature branch's commits:

```
git push origin shiny-new-feature
```

Here `origin` is the default name given to your remote repository on GitHub. You can see the remote repositories:

```
git remote -v
```

If you added the upstream repository as described above you will see something like:

```
origin  git@github.com:yourname/pydata-google-auth.git (fetch)
origin  git@github.com:yourname/pydata-google-auth.git (push)
upstream        git://github.com/pydata/pydata-google-auth.git (fetch)
upstream        git://github.com/pydata/pydata-google-auth.git (push)
```

Now your code is on GitHub, but it is not yet a part of the *pydata-google-auth* project. For that to happen, a pull request needs to be submitted on GitHub.

5.5.4 Review your code

When you're ready to ask for a code review, file a pull request. Before you do, once again make sure that you have followed all the guidelines outlined in this document regarding code style, tests, performance tests, and documentation. You should also double check your branch changes against the branch it was based on:

1. Navigate to your repository on GitHub – https://github.com/your-user-name/pydata-google-auth
2. Click on `Branches`
3. Click on the `Compare` button for your feature branch
4. Select the `base` and `compare` branches, if necessary. This will be `master` and `shiny-new-feature`, respectively.

5.5.5 Finally, make the pull request

If everything looks good, you are ready to make a pull request. A pull request is how code from a local repository becomes available to the GitHub community and can be looked at and eventually merged into the master version. This pull request and its associated changes will eventually be committed to the master branch and available in the next release. To submit a pull request:

1. Navigate to your repository on GitHub
2. Click on the `Pull Request` button
3. You can then click on `Commits` and `Files Changed` to make sure everything looks okay one last time
4. Write a description of your changes in the `Preview Discussion` tab
5. Click `Send Pull Request`.

This request then goes to the repository maintainers, and they will review the code. If you need to make more changes, you can make them in your branch, push them to GitHub, and the pull request will be automatically updated. Pushing them to GitHub again is done by:

```
git push -f origin shiny-new-feature
```

This will automatically update your pull request with the latest code and restart the CircleCI tests.

5.5.6 Delete your merged branch (optional)

Once your feature branch is accepted into upstream, you'll probably want to get rid of the branch. First, merge upstream master into your branch so git knows it is safe to delete your branch:

```
git fetch upstream
git checkout master
git merge upstream/master
```

Then you can just do:

```
git branch -d shiny-new-feature
```

Make sure you use a lower-case `-d`, or else git won't warn you if your feature branch has not actually been merged.

The branch will still exist on GitHub, so to delete it there do:

```
git push origin --delete shiny-new-feature
```

CHAPTER

SIX

CHANGELOG

6.1 1.8.2 / (2023-08-01)

- Ensure that the user credentials flow always gets a refresh token. (GH#72)

6.2 1.8.1 / (2023-07-10)

- If any exception occurs during Google Colab authentication, fallback to the Google Application Default Credentials flow. (GH#71)

6.3 1.8.0 / (2023-05-09)

- When running on Google Colab, try Colab-based authentication (`google.colab.auth.authenticate_user()`) before attempting the Google Application Default Credentials flow. (GH#68)

6.4 1.7.0 / (2023-02-07)

- Reissue of the library with the changes from 1.6.0 but with a new version number due to a conflict in releases.

6.5 1.6.0 / (2023-02-07)

- Adds decision logic to handle use cases where a user may not have the ability to log in via an Out of Band authentication flow. (GH#54)
- Also provides an OAuth page as part of the documentation.

6.6 1.5.0 / (2023-01-09)

- Adds ability to provide redirect uri. (GH#58)

6.7 1.4.0 / (2022-03-14)

- Default `use_local_webserver` to `True`. Google has deprecated the `use_local_webserver = False` "out of band" (copy-paste) flow. The `use_local_webserver = False` option is planned to stop working in October 2022.

6.8 1.3.0 / (2021-12-03)

- Adds support for Python 3.10. (GH#51)
- Fixes typo in documentation. (GH#44)

6.9 1.2.0 / (2021-04-21)

- Adds *`pydata_google_auth.load_service_account_credentials()`* function to get service account credentials from the specified JSON path. (GH#39)

6.10 1.1.0 / (2020-04-23)

- Try a range of ports between 8080 and 8090 when `use_local_webserver` is `True`. (GH#35)

6.11 1.0.0 / (2020-04-20)

- Mark package as 1.0, generally available.
- Update introduction with link to instructions on creating a Google Cloud project. (GH#18)

6.12 0.3.0 / (2020-02-04)

- Add `python -m pydata_google_auth` CLI for working with user credentials. (GH#28)

6.13 0.2.1 / (2019-12-12)

- Re-enable `auth_local_webserver` in `default` method. Show warning, rather than fallback to console.

6.14 0.2.0 / (2019-12-12)

- **Deprecate** `auth_local_webserver` argument in favor of `use_local_webserver` argument (GH#20).

6.14.1 New Features

- Adds *`pydata_google_auth.save_user_credentials()`* function to get user credentials and then save them to a specified JSON path. (GH#22)

6.14.2 Bug Fixes

- Update OAuth2 token endpoint to latest URI from Google. (GH#27)
- Don't raise error when the `APPDATA` environment variable isn't set on Windows. (GH#29)

6.15 0.1.3 / (2019-02-26)

6.15.1 Bug Fixes

- Respect the `dirname` and `filename` arguments to the *`ReadWriteCredentialsCache`* and *`WriteOnlyCredentialsCache`* constructors. (GH#16, GH#17)

6.16 0.1.2 / (2019-02-01)

6.16.1 Bug Fixes

- Don't write to the filesystem at module import time. This fixes an issue where the module could not be imported on systems where the file system is unwriteable. (GH#10, GH#11)

6.17 0.1.1 / (2018-10-26)

- Add LICENSE.txt to package manifest.
- Document privacy policy.

6.18 0.1.0 / (2018-10-23)

- Add `cache` module for configuring caching behaviors. (GH#1)
- Fork the pandas-gbq project and refactor out helpers for working with Google credentials.

CHAPTER

SEVEN

PRIVACY

This package is a PyData project and is subject to the NumFocus privacy policy. Your use of Google APIs with this module is subject to each API's respective terms of service.

7.1 Google account and user data

7.1.1 Accessing user data

The *pydata_google_auth* module accesses your Google user account, with the list of scopes that you specify. Depending on your specified list of scopes, the credentials returned by this library may provide access to other user data, such as your email address, Google Cloud Platform resources, Google Drive files, or Google Sheets.

7.1.2 Storing user data

By default, your credentials are stored by the *pydata_google_auth.cache.READ_WRITE* class to a local file, such as `~/.config/pydata`. All user data is stored on your local machine. **Use caution when using this library on a shared machine**.

7.1.3 Sharing user data

The pydata-google-auth library only communicates with Google APIs. No user data is shared with PyData, NumFocus, or any other servers.

7.2 Policies for application authors

Do not use the default client ID when using the pydata-google-auth library from an application, library, or tool. Per the Google User Data Policy, your application must accurately represent itself when authenticating to Google API servcies.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX