AGM Documentation

Release v0.1.0

Alex Wennerberg

Jun 02, 2020

Contents:

CHAPTER 1

AGM

AGM is an unofficial command line interface for Google APIs, written in Python. For example, if I wanted to get a list of all the items in my drive account, I would use the command:

```
agm drive files list --user myemail@gmail.com --fields "*"
```

AGM will parse this command and efficiently execute these requests. AGM works with all Google APIs and provides additional features which makes quickly processing data from Google's APIs simple and easy. It's great for debugging, G Suite administration, or simple task automation.

This tool is currenly in *Alpha*, so bugs are possible. Please report any issues to the issues tab on GitHub. If you'd like to contribute, please let me know or submit a pull request!

Developed and maintained by Alex Wennerberg

1.1 Documentation

More detailed documentation can be found on readthedocs!

1.1.1 Installation and Setup

Installation

AGM requires Python>=3.5. I haven't tested installation on Windows, but I plan on building it soon.

To install AGM via pip, run this command:

```
pip install --user agm
```

If you don't have pip installed, follow this guide. Depending on your Python environment, you may need to run `pip3` instead of `pip`. If you haven't set up a Python environment on your machine and are using a Linux-based system, you may need to add the line `export PATH=$PATH":$HOME/.local/bin"` to your `~/.bashrc` file.

Setup

AGM can be authenticated in two ways, via service account or through individual accounts via OAuth2. Using a service account is useful if you're a developer or G Suite administrator, whereas individual authentication is useful if you're acting on behalf of a single account, a consumer Google account or a list of consumer Google accounts.

Service account authentication

This will grant agm access to all users in a domain.

First, follow Google's guide for Using OAuth 2.0 for Server to Server Applications. Put this JSON keyfile into `~/.agm/` (preferred) or pass it into AGM with the `--keyfile` flag. Beware this key is tremendously powerful. Keep it secret and safe and don't share it with anyone! Be mindful of the scopes you are using and refurnish a new key from time to time.

AGM does support p12 keys, just make sure that the filename of the key is renamed to [service account email].p12

Make sure any APIs you wanna use are enabled in the project associated with your service account and that the scopes are added to your key's client ID as described in these docs

Individual OAuth2 authentication

This will grant AGM access to individual user accounts.

Navigate to the Google API credentials page.

If you haven't set up this project yet, you will need to configure your OAuth2 consent screen. Fill out the Application Name field (I recommend "AGM") and leave all the other fields as defaults.

Select "Create Crenedentials" and "OAuth Client ID". Select "Other" for application type and give your client a name (I recommend AGM). On the credentials page, download the client secret and save it into `~/.agm/client_secret.json`. This client secret will serve as the indentifier of your AGM installation, you can freely use the same client secret to authenticate many individual Google accounts.

In order to run the oauth authentication, use the following command:

```
agm --run-oauth --user [emaill_to_authenticate] --scopes [scopes to grant]*
```

You may receive a warning at this point. Click "advanced" and proceed anyway.

*See the list of API scopes and decide on the scopes that you need for your purpose. You can always change these by running this command again with different scopes.

Authenticated credentials will be stored in `~/.agm/oauth_credentials`. These keys are sensitive, so keep them safe and regularly delete any old credentials you don't need anymore.

You will now be able to make AGM requests on behalf of this user, by specifying them with the `--user` parameter.

1.1.2 Using AGM

Basic Usage

```
usage: agm [-h] [-v] [-u [USER [USER ...]]] [--outfile OUTFILE] [-d]
           [--version] [--scopes SCOPES [SCOPES ...]] [--keyfile KEYFILE] [-V]
           [--run_oauth] [--auth_info]
           [command [command ...]]


 _______ _______ __    __
|   _   |       |  |_|  |
|  |_|  |    ___|       |
|       |   | __|       |
|       |   ||  |       |
|   _   |   |_| | ||_|| |
|__| |__|_______|_|   |_|
version: 0.1.0

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         longer output

API:
  Call the Google API or get docs with agm [service] [resource] [method]

  command               The name of the service to call e.g. drive files list.
  -u [USER [USER ...]], --user [USER [USER ...]], --users [USER [USER ...]]
                        The impersonated user(s).
  --outfile OUTFILE     Send output to a given file.
  -d, --docs            Open the documentation for this function.
  --version             specify api version, ie 'v2'. Optional
  --scopes SCOPES [SCOPES ...]
                        API scopes to give the key. Optional and usually
                        unnecessary. If not specified, AGM will try all
                        possible scopes that could authenticate a command
                        until it finds one that works. You can drop everything
                        before the last slash in the url, eg just use
                        gmail.readonly
  --keyfile KEYFILE     Optional. specify the keyfile to authenticate with. By
                        default, AGMlooks for a .json file inside ~/.agm

Tools:
  AGM tools

  -V                    AGM version
  --run_oauth           Authenticate an individual Google account and store
                        the credentials.Requires scopes and user to be set.
  --auth_info            Print information about authenticated keys
```

AGM maps commands to the associated Google API. Not all APIs have been tested, so please report any issues to the GitHub Issues page. Separate the API name (e.g. gmail) from the resources (e.g. user messages) from the method, like so:

```
agm gmail users messages list --user myemail@gmail.com --maxResults 100 --userId me
```

This will return a json with a summary of the request and and the response from the server. If the request failed, the error message will be in the "error" field. You can provide a list for any parameter and AGM will iterate over that list.

For example, if I wanted to get information about a list of files,

NOTE: If you want to get a list of items and specify a field selection, make sure to specify "nextPageToken" in your fields query

```
agm drive files get --user myemail@gmail.com --fileId abc def ghi
```

And AGM will iterate over that list in order to get information about each file. If you provide multiple values for multiple parameters, AGM will iterate over each list together. AGM uses batch requests and multithreading for high performance, so AGM is especially suitable for very large, long-running tasks that involve many API requests.

AGM will automatically convert flags into data to send to the request body. For example if I want to create a file in drive, I can send the body directly as a key-value dictionary, such as

```
agm drive files create --user myemail@gmail.com --body "{'name': 'myfile'}"
```

This is a bit cumbersome, so AGM allows you to pass flags, and will determine whether or not they should be in the body of the request, like so:

```
agm drive files create --user myemail@gmail.com --name myfile
```

For documentation on a command, run the command with the `-d` or `--docs` flag. This flag can be ommitted if you want information about a service or resource. Just typing `agm` will list all of the available APIs.

If `--outfile` is ommitted, output will be printed to the console.

Logs are in ~/.agm/logs. Check them for debugging or if you need a record of recent actions performed by AGM.

Piping input

You can pipe input into AGM. For our previous command, if we wanted to get information about a large number of files, we could save these files to a text document, files.txt, and then pipe that file into AGM:

```
cat files.txt | agm drive files get --user myemail@gmail.com --fileId
```

AGM will also accept a json string as piped input of the format:

```
{
  "parameter": "value"
}
```

Note that performance will degrade if you try and pipe in an extremely large amount of input. This is an open issue that I am looking to resolve.

Advanced Usage

AGM is built to interact with other Unix command line tools, such as [jq](https://stedolan.github.io/jq/)

See the [examples](https://github.com/Cloudbakers/agm/tree/master/examples) folder for some examples of more complex actions

1.1.3 Development

AGM tries to follow the Unix principle of Do One Thing and Do It Welll. This means things like parsing JSON output, converting to CSVs or uploading results to a Google Sheet, etc are not part of AGM. There are other great tools that you can use with AGM to accomplish these tasks. I want to keep this codebase as simple and focused as possible.

I also don’t want to build out functionality that caters to a specific API, ie some extra Google Drive features. The behavior of AGM should be as transparent as possible and deviate from what it would look like to make a curl request to the Google API as little as possible. There are very minor changes I make (such as setting a default fields value of *), but these changes should be limited in scope and easy to override.

I would love help with refactoring, documentation, QA, and adding tests.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search