# acd_cli Documentation

***Release 0.3.1***

**yadayada**

February 20, 2016

Contents

Version 0.3.1

Contents:

CHAPTER 1

# Setting up acd_cli

Check which Python 3 version is installed on your system, e.g. by running

```
python3 -V
```

If it is Python 3.2.3, 3.3.0 or 3.3.1, you need to upgrade to a higher minor version.

You may now proceed to install using PIP, your Arch package manager or build Debian/RedHat packages.

## 1.1 Installation with PIP

If you are new to Python, worried about dependencies or about possibly messing up your system, create and activate virtualenv like so:

```
cd /parent/path/to/your/new/virtualenv
virtualenv acdcli
source acdcli/bin/activate
```

You are now safe to install and test acd_cli. When you are finished, the environment can be disabled by simply closing your shell or running `deactivate`.

Please check which pip command is appropriate for Python 3 packages in your environment. I will be using 'pip3' as superuser in the examples.

The recommended and most up-to-date way is to directly install the master branch from GitHub.

```
pip3 install --upgrade git+https://github.com/yadayada/acd_cli.git
```

The easiest way is to directly install from PyPI.

```
pip3 install --upgrade --pre acdcli
```

### 1.1.1 PIP Errors

A version incompatibility may arise with PIP when upgrading the requests package. PIP will throw the following error:

```
ImportError: cannot import name 'IncompleteRead'
```

Run these commands to fix it:

```
apt-get remove python3-pip
easy_install3 pip
```

This will remove the distribution's pip3 package and replace it with a version that is compatible with the newer requests package.

## 1.2 Installation on Arch/Debian/RedHat

### 1.2.1 Arch Linux

There are two packages for Arch Linux in the AUR, acd_cli-git, which is linked to the master branch of the GitHub repository, and acd_cli, which is linked to the PyPI release.

### 1.2.2 Building deb/rpm packages

You will need to have fpm installed to build packages.

There is a Makefile that includes commands to build Debian packages (`make deb`) or RedHat packages (`make rpm`). It will also build the required requests-toolbelt package. fpm may also be able to build packages for other distributions or operating systems.

## 1.3 Environment Variables

### 1.3.1 Cache Path and Settings Path

You will find the current path settings in the output of `acd_cli -v init`.

The cache path is where acd_cli stores OAuth data, the node cache, logs etc. You may override the cache path by setting the `ACD_CLI_CACHE_PATH` environment variable.

### 1.3.2 Proxy support

Requests supports HTTP(S) proxies via environment variables. Since all connections to Amazon Cloud Drive are using HTTPS, you need to set the variable `HTTPS_PROXY`. The following example shows how to do that in a bash-compatible environment.

```
export HTTPS_PROXY="https://user:pass@1.2.3.4:8080/"
```

### 1.3.3 Locale

If you need non-ASCII file/directory names, please check that your system's locale is set correctly.

## 1.4 Dependencies

### 1.4.1 FUSE

For the mounting feature, fuse >= 2.6 is needed according to fusepy. On a Debian-based distribution, the package should be named simply 'fuse'.

### 1.4.2 Python Packages

Under normal circumstances, it should not be necessary to install the dependencies manually.

- appdirs
- colorama
- dateutils (recommended)
- requests >= 2.1.0
- requests-toolbelt (recommended)
- sqlalchemy

Recommended packages are not strictly necessary; but they will be preferred to workarounds (in the case of dateutils) and bundled modules (requests-toolbelt).

If you want to the dependencies using your distribution's packaging system and are using a distro based on Debian 'jessie', the necessary packages are `python3-appdirs python3-colorama python3-dateutil python3-requests python3-sqlalchemy`.

## 1.5 Uninstalling

Please run `acd_cli delete-everything` first to delete your authentication and node data in the cache path. Then, use pip to uninstall

```
pip3 uninstall acdcli
```

Then, revoke the permission for `acd_cli_oa` to access your cloud drive in your Amazon profile, more precisely at https://www.amazon.com/ap/adam.

CHAPTER 2

# Authorization

Before you can use the program, you will have to complete the OAuth procedure with Amazon. There is a fast and simple way and a secure way.

## 2.1 Simple (Appspot)

You will not have to prepare anything to initiate this authorization method, just run, for example, `acd_cli init`.

A browser (tab) will open and you will be asked to log into your Amazon account or grant access for 'acd_cli_oa'. Signing in or clicking on 'Continue' will download a JSON file named `oauth_data`, which must be placed in the cache directory displayed on screen (e.g. `/home/<USER>/.cache/acd_cli`).

You may view the source code of the Appspot app that is used to handle the server part of the OAuth procedure at https://tensile-runway-92512.appspot.com/src.

## 2.2 Advanced Users (Security Profile)

You must create a security profile and have it whitelisted. Have a look at Amazon's ACD getting started guide. Select all permissions for your security profile and add a redirect URL to `http://localhost`.

Put your own security profile data in a file called `client_data` in the cache directory and have it adhere to the following form.

```
{
    "CLIENT_ID": "amzn1.application-oa2-client.0123456789abcdef0123456789abcdef",
    "CLIENT_SECRET": "0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef"
}
```

You may now run `acd_cli -v init`. The authentication procedure is similar to the one above. A browser (tab) will be opened and you will be asked to log in. Unless you have a local webserver running on port 80, you will be redirected to your browser's error page. Just copy the URL (e.g. `http://localhost/?code=AbCdEfGhIjKlMnOpQrSt&scope=clouddrive%3Aread_all+clouddrive%3Awrite`) into the console.

## 2.3 Changing authorization methods

If you want to change between authorization methods, go to your cache path (it is stated in the output of `acd_cli -v init`) and delete the file `oauth_data` and, if it exists, `client_data`.

CHAPTER 3

# Usage

acd_cli may be invoked as `acd_cli` or `acdcli`.

Most actions need the node cache to be initialized and up-to-date, so please run a sync. A sync will fetch the changes since the last sync or the full node list if the cache is empty.

The following actions are built in

```
sync (s)            refresh node list cache; necessary for many actions
clear-cache (cc)    clear node cache [offline operation]

tree (t)            print directory tree [offline operation]
children (ls)       list a folder's children [offline operation]

find (f)            find nodes by name [offline operation] [case insensitive]
find-md5 (fm)       find files by MD5 hash [offline operation]
find-regex (fr)     find nodes by regular expression [offline operation] [case insensitive]

upload (ul)         file and directory upload to a remote destination
overwrite (ov)      overwrite file A [remote] with content of file B [local]
stream (st)         upload the standard input stream to a file
download (dl)       download a remote folder or file; will skip existing local files
cat                 output a file to the standard output stream

create (c, mkdir)   create folder using an absolute path

list-trash (lt)     list trashed nodes [offline operation]
trash (rm)          move node to trash
restore (re)        restore node from trash

move (mv)           move node A into folder B
rename (rn)         rename a node

resolve (rs)        resolve a path to a node ID [offline operation]

usage (u)           show drive usage data
quota (q)           show drive quota [raw JSON]
metadata (m)        print a node's metadata [raw JSON]

mount               mount the cloud drive at a local directory
umount              unmount cloud drive(s)
```

Please run `acd_cli --help` to get a current list of the available actions. A list of further arguments of an action and their order can be printed by calling `acd_cli [action] --help`.

Most node arguments may be specified as a 22 character ID or a UNIX-style path. Trashed nodes' paths might not be able to be resolved correctly; use their ID instead.

There are more detailed instructions for file transfer actions, find actions and FUSE documentation.

Logs will automatically be saved into the cache directory.

## 3.1 Global Flags/Parameters

`--verbose` (`-v`) and `--debug` (`-d`) will print additional messages to standard error.

`--no-log` (`-nl`) will disable the automatic logging feature that saves log files to the cache directory.

`--color` will set the coloring mode according to the specified argument (`auto`, `never` or `always`). Coloring is turned off by default; it is used for file/folder listings.

`--check` (`-c`) sets the start-up database integrity check mode. The default is to perform a `full` check. Setting the check to `quick` or `none` may speed up the initialization for large databases.

`--utf` (`-u`) will force the output to be encoded in UTF-8, regardless of the system's settings.

## 3.2 Exit Status

When the script is done running, its exit status can be checked for flags. If no error occurs, the exit status will be 0. Possible flag values are:

| flag | value |
|---|---|
| general error | 1 |
| argument error | 2 |
| failed file transfer | 8 |
| upload timeout | 16 |
| hash mismatch | 32 |
| error creating folder | 64 |
| file size mismatch | 128 |
| cache outdated | 256 |
| remote duplicate | 512 |
| duplicate inode | 1024 |
| name collision | 2048 |
| error deleting source file | 4096 |

If multiple errors occur, their values will be compounded by a binary OR operation.

CHAPTER 4

# File transfer

acd_cli offers multi-file transfer actions - upload and download - and single-file transfer actions - overwrite, stream and cat.

Multi-file transfers can be done with concurrent connections by specifying the argument `-x NUM`. If remote folder hierarchies or local directory hierarchies need to be created, this will be done prior to the file transfers.

## 4.1 Actions

### 4.1.1 `upload`

The upload action will upload files or recursively upload directories. Existing files will not be changed, normally.

Syntax:

```
acdcli upload /local/path [/local/next_path [...]] /remote/path
```

If the `--overwrite` (`-o`) argument is specified, a remote file will be updated if a) the local file's modification time is higher or b) the local file's creation time is higher and the file size is different. The `--force` (`-f`) argument can be used to force overwrite.

**Hint:** When uploading large files (>10GiB), a warning about a timeout may be displayed. You then need to wait a few minutes, sync and manually check if the file was uploaded correctly.

### 4.1.2 `overwrite`

The upload action overwrites the content of a remote file with a local file.

Syntax:

```
acdcli overwrite /local/path /remote/path
```

### 4.1.3 `download`

The download action can download a single file or recursively download a directory. If a file already exists locally, it will not be overwritten.

Syntax:

```
acdcli download /remote/path [/local/path]
```

If the local path is omitted, the destination path will be the current working directory.

### 4.1.4 stream

This action will upload the standard input stream to a file.

Syntax:

```
some_process | acdcli stream file_name /remote/path
```

If the `--overwrite` (`-o`) argument is specified, the remote file will be overwritten if it exists.

### 4.1.5 cat

This action outputs the content of a file to standard output.

## 4.2 Abort/Resume

Incomplete file downloads will be resumed automatically. Aborted file uploads are not resumable at the moment.

Folder or directory hierarchies that were created for a transfer do not need to be recreated when resuming a transfer.

## 4.3 Retry

Failed upload, download and overwrite actions allow retries on error by specifying the `--max-retries|-r` argument, e.g. `acd_cli <ACTION> -r MAX_RETRIES`.

## 4.4 Exclusion

Files may be excluded from upload or download by regex on their name or by file ending. Additionally, paths can be excluded from upload. Regexes and file endings are case-insensitive.

It is possible to specify multiple exclusion arguments of the same kind.

## 4.5 Deduplication

Server-side deduplication prevents completely uploaded files from being saved as a node if another file with the same MD5 checksum already exists. acd_cli can prevent uploading duplicates by checking local files' sizes and MD5s. Empty files are never regarded duplicates.

CHAPTER 5

# Finding nodes

The find actions will search for normal (active) and trashed nodes and list them.

## 5.1 find

The find action will perform a case-insensitive search for files and folders that include the name or name segment given as argument, so e.g. `acdcli find foo` will find "foo" , "Foobar", etc.

## 5.2 find-md5

find-md5 will search for files that match the MD5 hash given. The location of a local file may be determined like so:

```
acdcli find-md5 `md5sum local/file | cut -d" " -f1`
```

## 5.3 find-regex

find-regex searches for the specified regex in nodes' names.

CHAPTER 6

# FUSE module

## 6.1 Status

The FUSE support is still in its early stage and may be (prone to bugs). acd_cli's FUSE module has the following filesystem features implemented:

| Feature | Working |
|---|---|
| Basic operations | |
| List directory | |
| Read | |
| Write | [1] |
| Rename | |
| Move | |
| Trashing | |
| OS-level trashing | [2] |
| View trash | |
| Misc | |
| Automatic sync | |
| ctime/mtime update | |
| Custom permissions | |
| Hard links | partially [3] |
| Symbolic links | [4] |

## 6.2 Usage

The command to mount the (root of the) cloud drive to the empty directory `path/to/mountpoint` is

```
acd_cli mount path/to/mountpoint
```

A cloud drive folder may be mounted similarly, by

```
acd_cli mount --modules="subdir,subdir=/folder" path/to/mountpoint
```

Unmounting is usually achieved by the following command

[1] partial writes are not possible (i.e. writes at random offsets)
[2] restoring might not work
[3] manually created hard links will be displayed, but it is discouraged to use them
[4] soft links are not part of the ACD API

```
fusermount -u path/to/mountpoint
```

If the mount is busy, Linux users can use the `--lazy` (`-z`) flag. There exists a convenience action `acd_cli umount` that unmounts all ACDFuse mounts on Linux and Mac OS.

### 6.2.1 Mount options

For further information on the most of the options below, see your mount.fuse man page.

To convert the node's standard character set (UTF-8) to the system locale, the modules argument may be used, e.g. `--modules="iconv,to_code=CHARSET"`.

**--allow-other, -ao** allow all users to access the mountpoint (may need extra configuration)

**--allow-root, -ar** allow the root user to access the mountpoint (may need extra configuration)

**--foreground, -fg** do not detach process until filesystem is destroyed (blocks)

**--interval INT, -i INT** set the node cache sync (refresh) interval to INT seconds

**--nlinks, -n** calculate the number of links for folders (slower)

**--nonempty, -ne** allow mounting to a non-empty mount point

**--read-only, -ro** disallow write operations (does not affect cache refresh)

**--single-threaded, -st** disallow multi-threaded FUSE operations

### 6.2.2 Automatic remount

Linux users may use the systemd service file from the assets directory to have the clouddrive automatically remounted on login. Alternative ways are to add a crontab entry using the `@reboot` keyword or to add an fstab entry like so:

```
acdmount    /mount/point    fuse    defaults    0   0
```

For this to work, an executable shell script /usr/bin/acdmount must be created

```
#!/bin/bash

acd_cli mount $1
```

Please make sure your network connection is up before these commands are executed or the mount will fail.

### 6.2.3 Library Path

If you want or need to override the standard libfuse path, you may set the environment variable *LIBFUSE_PATH* to the full path of libfuse, e.g.

```
export LIBFUSE_PATH="/lib/x86_64-linux-gnu/libfuse.so.2"
```

This is particularly helpful if the libfuse library is properly installed, but not found.

### 6.2.4 Deleting Nodes

"Deleting" directories or files from the file system will only trash them in your cloud drive. Calling rmdir on a directory will always move it into the trash, even if it is not empty.

### 6.2.5 Logging

For debugging purposes, the recommended command to run is

```
acd_cli -d -nl mount -i0 -fg path/to/mountpoint
```

That command will disable the automatic refresh (i.e. sync) of the node cache (*-i0*) and disable detaching from the console.

CHAPTER 7

# Contributing guidelines

## 7.1 Using the Issue Tracker

The issue tracker is not a forum! This does not mean there is no need for good etiquette, but that you should not post unnecessary information. Each reply will cause a notification to be sent to all of the issue's participants and some of them might consider it spam.

For minor corrections or additions, try to update your posts rather than writing a new reply. Use strike-through markdown for corrections and put updates at the bottom of your original post.

+1ing an issue or "me, too" replies will not get anything done faster.

### 7.1.1 Adding Issues

If you have a question, please read the documentation and search the issue tracker. If you still have a question, please consider using the Gitter chat or sending an e-mail to acd_cli@mail.com instead of opening an issue.

If you absolutely must open an issue, check that you are using the latest master commit and there is no existing issue that fits your problem (including closed and unresolved issues). Try to reproduce the issue on another machine or ideally on another operating system, if possible.

Please provide as much possibly relevant information as you can. This should at least contain:

- your operating system and Python version, e.g. as determined by `python3 -c 'import platform as p; print("%s\n%s" % (p.python_version(), p.platform()))'`
- the command/s you used
- what happened
- what you think should have happened instead (and maybe give a reason)

You might find the `--verbose` and, to a lesser extent, `--debug` flags helpful.

Use code block markup for console output, log messages, etc.

## 7.2 Code

There are no real programming guidelines as of yet. Please use function annotations for typing like specified in PEP 3107 and, to stay 3.2-compliant, stringified PEP 484 type hints where appropriate. The limit on line length is 100 characters.

It is a generally a good idea to explicitly announce that you are working on an issue.

Please squash your commits and add yourself to the contributors list before making a pull request.

Have a look at Github's general guide how to contribute. It is not necessary to create a feature branch, i.e. you may commit to the master branch.

There is also a TODO list of some of the open tasks.

## 7.3 Donations

You might also want to consider making a donation to further the development of acd_cli.

CHAPTER 8

# Contributors

Thanks to

- chrisidefix for adding the find-md5 action and forcing me to create a proper package and use PyPI
- msh100 for adding proxy documentation and updating the oauth scope
- hansendc for revamping the usage report
- legnaleurc for adding the find-regex action
- Timdawson264 for fixing st_nlinks in the FUSE node stat
- Lorentz83 for creating a bash completion script
- kylemanna for adding a systemd service file

Also thanks to

- fibersnet for pointing out a possible deadlock in ACDFuse.
- and everyone else who I forgot to mention

CHAPTER 9

# Frequently Asked Questions

## 9.1 Why Did I Get an UnicodeEncodeError?

If you encounter Unicode problems, check that your locale is set correctly. Alternatively, you may use the `--utf` argument to force acd_cli to use UTF-8 output encoding regardless of your console's current encoding.

Windows users may import the provided reg file (assets/win_codepage.reg), tested with Windows 8.1, to set the command line interface encoding to cp65001.

## 9.2 What Is acd_cli's Installation Path?

On unixoid operating systems the acd_cli script may be located by running `which acd_cli` or, if that does not yield a result, by executing `pip3 show -f acdcli`.

## 9.3 Where Does acd_cli Store its Cache and Settings?

You can see which paths are used in the log output of `acd_cli -v init`.

## 9.4 How Do I Pass a Node ID Starting with – (dash/minus/hyphen)?

Precede the node ID by two minuses and a space to have it be interpreted as parameter and not as an argument, e.g. `-- -AbCdEfGhIjKlMnOpQr012`.

## 9.5 Can I Share or Delete Files/Folders?

No. It is not possible to share or delete using the Cloud Drive API. Please do it manually using the Web interface.

## 9.6 How Do I Share Directories from ACDFuse with Samba?

By default, only the user that originally mounted the FUSE filesystem has access permissions. To lift this restriction, run the `mount` command with the `--allow-other` option. You may need to edit your system's setting before being able to use this mount option, e.g. in /etc/fuse.conf.

## 9.7 Do Transfer Speeds Vary Depending on Geolocation?

Amazon may be throttling users not located in the U.S. To quote the Terms of Use,

> The Service is offered in the United States. We may restrict access from other locations. There may be limits on the types of content you can store and share using the Service, such as file types we don't support, and on the number or type of devices you can use to access the Service. We may impose other restrictions on use of the Service.

CHAPTER 10

# Ancient History

## 10.1 0.1.3

- plugin mechanism added
- OAuth now via Appspot; security profile no longer necessary
- back-off algorithm for API requests implemented

## 10.2 0.1.2

**new:**

- overwriting of files
- recursive upload/download
- hashing of downloaded files
- clear-cache action

**fixes:**

- remove-child accepted status code
- fix for upload of files with Unicode characters

**other:**

- changed database schema

CHAPTER 11

# Development

Contents:

## 11.1 acdcli package

### 11.1.1 Subpackages

#### acdcli.api package

##### Submodules

##### acdcli.api.account module

ACD account information

**class** acdcli.api.account.**AccountMixin**
Bases: object

**fs_sizes**() → tuple

**Returns tuple** total and free space

**get_account_info**() → dict
Gets account status [ACTIVE, ...?] and terms of use version.

**get_account_usage**() → str

**get_quota**() → dict

##### acdcli.api.backoff_req module

**class** acdcli.api.backoff_req.**BackOffRequest**(*auth_callback: 'requests.auth.AuthBase'*)
Bases: object

Wrapper for requests that implements timed back-off algorithm https://developer.amazon.com/public/apis/experience/cloud-drive/content/best-practices Caution: this catches all connection errors and may stall for a long time. It is necessary to init this module before use.

**__init__**(*auth_callback: 'requests.auth.AuthBase'*)

**Parameters** **auth_callback** – callable object that attaches auth info to a request

**delete**(*url, acc_codes=[200], **kwargs*) → requests.models.Response

**get**(*url, acc_codes=[200], **kwargs*) → requests.models.Response

**paginated_get**(*url: str, params: dict=None*) → 'List[dict]'
Gets node list in segments of 200.

**patch**(*url, acc_codes=[200], **kwargs*) → requests.models.Response

**post**(*url, acc_codes=[200], **kwargs*) → requests.models.Response

**put**(*url, acc_codes=[200], **kwargs*) → requests.models.Response

acdcli.api.backoff_req.**CONN_TIMEOUT = 30**
timeout for establishing a connection

acdcli.api.backoff_req.**IDLE_TIMEOUT = 60**
read timeout

acdcli.api.backoff_req.**REQUESTS_TIMEOUT = (30, 60)**
http://docs.python-requests.org/en/latest/user/advanced/#timeouts

### acdcli.api.client module

**class** acdcli.api.client.**ACDClient**(*path=''*)
Bases: *acdcli.api.account.AccountMixin*, *acdcli.api.content.ContentMixin*, *acdcli.api.metadata.MetadataMixin*, *acdcli.api.trash.TrashMixin*

Provides a client to the Amazon Cloud Drive RESTful interface.

**__init__**(*path=''*)
Initializes OAuth and endpoints.

**content_url**

**metadata_url**

acdcli.api.client.**ENDPOINT_VAL_TIME = 259200**
number of seconds for endpoint validity (3 days)

### acdcli.api.common module

**exception** acdcli.api.common.**RequestError**(*status_code: int, msg: str*)
Bases: Exception

Catch-all exception class for various connection and ACD server errors.

**class CODE**
Bases: object

**CONN_EXCEPTION = 1000**

**FAILED_SUBREQUEST = 1002**

**INCOMPLETE_RESULT = 1003**

**INVALID_TOKEN = 1005**

**REFRESH_FAILED = 1004**

RequestError.**__init__**(*status_code: int, msg: str*)

RequestError.**codes = <lookup 'status_codes'>**

acdcli.api.common.**catch_conn_exception**(*func*)
Request connection exception decorator :raises RequestError

acdcli.api.common.**is_valid_id**(*id: str*) → bool

#### acdcli.api.content module

acdcli.api.content.**CHUNK_MAX_RETRY = 5**
retry limit for failed chunk

acdcli.api.content.**CHUNK_SIZE = 524288000**
download chunk size

**class** acdcli.api.content.**ContentMixin**
Bases: object

Implements content portion of the ACD API.

**chunked_download**(**args*, ***kwargs*)

**clear_file**(*node_id: str*) → dict
Clears a file's content by overwriting it with an empty BytesIO.

**Parameters** **node_id** – valid file node ID

**create_file**(*file_name: str*, *parent: str=None*) → dict

**create_folder**(*name: str*, *parent=None*) → dict

**download_chunk**(*node_id: str*, *offset: int*, *length: int*, ***kwargs*) → bytearray
Load a file chunk into memory.

**Parameters** **length** – the length of the download chunk

**download_file**(*node_id: str*, *basename: str*, *dirname: str=None*, ***kwargs*)
Deals with download preparation, download with *chunked_download()* and finish. Calls callbacks while fast forwarding through incomplete file (if existent). Will not check for existing file prior to download and overwrite existing file on finish.

**Parameters**

- **dirname** – a valid local directory name, or cwd if None
- **basename** – a valid file name
- **kwargs** –
  - length: the total length of the file
  - write_callbacks (list[function]): passed on to *chunked_download()*
  - resume (bool=True): whether to resume if partial file exists

**download_thumbnail**(*node_id: str*, *file_name: str*, *max_dim=128*)
Download a movie's or picture's thumbnail into a file. Officially supports the image formats JPEG, BMP, PNG, TIFF, some RAW formats and the video formats MP4, QuickTime, AVI, MTS, MPEG, ASF, WMV, FLV, OGG. See http://www.amazon.com/gp/help/customer/display.html?nodeId=201634590 Additionally supports MKV.

**Parameters** **max_dim** – maximum width or height of the resized image/video thumbnail

**overwrite_file**(*node_id: str*, *file_name: str*, *read_callbacks: list=None*, *deduplication=False*) → dict

**overwrite_stream**(*stream*, *node_id: str*, *read_callbacks: list=None*) → dict
Overwrite content of node with ID *node_id* with content of *stream*.

**Parameters** **stream** – readable object

**response_chunk**(*node_id: str*, *offset: int*, *length: int*, ***kwargs*) → requests.models.Response

**upload_file**(*file_name: str*, *parent: str=None*, *read_callbacks=None*, *deduplication=False*) → dict

**upload_stream**(*stream*, *file_name: str*, *parent: str=None*, *read_callbacks=None*, *deduplication=False*) → dict

**Parameters**

- **stream** – readable object
- **parent** – parent node id, defaults to root node if None

acdcli.api.content.**FS_RW_CHUNK_SZ = 131072**
basic chunk size for file system r/w operations

acdcli.api.content.**PARTIAL_SUFFIX = '.__incomplete'**
suffix (file ending) for incomplete files

#### acdcli.api.metadata module

Node metadata operations

acdcli.api.metadata.**ChangeSet**
alias of Changes

**class** acdcli.api.metadata.**MetadataMixin**
Bases: object

**add_child**(*parent_id: str*, *child_id: str*) → dict
Adds node with ID *child_id* to folder with ID *parent_id*.

**Returns** updated child node dict

**add_property**(*node_id: str*, *owner_id: str*, *key: str*, *value: str*) → dict
Adds or overwrites *key* property with *content*. Maximum number of keys per owner is 10.

**Parameters** **value** – string of length <= 500

**Raises** RequestError: 404, <UnknownOperationException/> if owner is empty RequestError: 400, {...} if maximum of allowed properties is reached

**Returns dict** {'key': '<KEY>', 'location': '<NODE_ADDRESS>/properties/<OWNER_ID/<KEY>',

'value': '<VALUE>'}

**delete_properties**(*node_id: str*, *owner_id: str*)
Deletes all of the owner's properties. Uses multiple requests.

**delete_property**(*node_id: str*, *owner_id: str*, *key: str*)
Deletes *key* property from node with ID *node_id*.

**get_asset_list**() → list

**get_changes**(*checkpoint=''*, *include_purged=False*) → 'Generator[ChangeSet]'
Generates a ChangeSet for each checkpoint in changes response. See https://developer.amazon.com/public/apis/experience/cloud-drive/content/changes.

**get_file_list**() → list

**get_folder_list**() → list

**get_metadata**(*node_id: str*, *assets=False*, *temp_link=True*) → dict
Gets a node's metadata.

**get_node_list**(**params*) → list

**Parameters** **params** – may include tempLink='True'

**get_owner_id**()
Provisional function for retrieving the security profile's name, a.k.a. owner id.

**get_root_id**() → str
Gets the ID of the root node

**Returns** the topmost folder id

**get_trashed_files**() → list

**get_trashed_folders**() → list

**list_children**(*node_id: str*) → list

**list_properties**(*node_id: str*, *owner_id: str*) → dict
This will always return an empty dict if the accessor is not the owner. :param _sphinx_paramlinks_acdcli.api.metadata.MetadataMixin.list_properties.owner_id: owner ID (return status 404 if empty)

**move_node**(*node_id: str*, *parent_id: str*) → dict

**move_node_from**(*node_id: str*, *old_parent_id: str*, *new_parent_id: str*) → dict
Moves node with given ID from old parent to new parent. Not tested with multi-parent nodes.

**Returns** changed node dict

**remove_child**(*parent_id: str*, *child_id: str*) → dict

**Returns** updated child node dict

**rename_node**(*node_id: str*, *new_name: str*) → dict

**set_available**(*node_id: str*) → dict
Sets node status from 'PENDING' to 'AVAILABLE'.

**update_metadata**(*node_id: str*, *properties: dict*) → dict
Update a node's properties like name, description, status, parents, ...

### acdcli.api.oauth module

**class** acdcli.api.oauth.**AppspotOAuthHandler**(*path*)
Bases: *acdcli.api.oauth.OAuthHandler*

**APPSPOT_URL = 'https://tensile-runway-92512.appspot.com/'**

**__init__**(*path*)

**check_oauth_file_exists**()
Checks for existence of oauth token file and instructs user to visit the Appspot page if it was not found.

**Raises** FileNotFoundError if oauth file was not placed into cache directory

**refresh_auth_token**()

**Raises** RequestError

**class** acdcli.api.oauth.**LocalOAuthHandler**(*path*)
Bases: *acdcli.api.oauth.OAuthHandler*

A local OAuth handler that works with a whitelisted security profile. The profile must not be created prior to June 2015. Profiles created prior to this month are not able to use the new scope "clouddrive:read_all" that replaces "clouddrive:read". https://developer.amazon.com/public/apis/experience/cloud-drive/content/getting-started

**AMAZON_OA_LOGIN_URL = 'https://amazon.com/ap/oa'**

**AMAZON_OA_TOKEN_URL = 'https://api.amazon.com/auth/o2/token'**

**CLIENT_DATA_FILE = 'client_data'**

**REDIRECT_URI = 'http://localhost'**

**__init__**(*path*)

**check_oauth_file_exists**()

**Raises** Exception

**load_client_data**()

**Raises** IOError if client data file was not found

**Raises** KeyError if client data file has missing key(s)

**refresh_auth_token**()

**Raises** RequestError

**class** acdcli.api.oauth.**OAuthHandler**(*path*)
Bases: requests.auth.AuthBase

**class KEYS**
Bases: object

**ACC_TOKEN = 'access_token'**

**EXP_IN = 'expires_in'**

**EXP_TIME = 'exp_time'**

**REDIRECT_URI = 'redirect_uri'**

**REFR_TOKEN = 'refresh_token'**

OAuthHandler.**OAUTH_DATA_FILE = 'oauth_data'**

OAuthHandler.**__init__**(*path*)

OAuthHandler.**check_oauth_file_exists**()
Checks for OAuth file existence and one-time initialize if necessary. Throws on error.

OAuthHandler.**exp_time**

OAuthHandler.**get_access_token_info**() → dict

**Returns**

int exp: expiration time in sec, str aud: client id user_id, app_id, iat (exp time)

OAuthHandler.**get_auth_token**(*reload=True*) → str
Gets current access token, refreshes if necessary.

**Parameters** **reload** – whether the oauth token file should be reloaded (external update)

OAuthHandler.**load_oauth_data**()
Loads oauth data file, validate and add expiration time if necessary

OAuthHandler.**refresh_auth_token**()
Fetches a new access token using the refresh token.

OAuthHandler.**treat_auth_token**(*time_: float*)
Adds expiration time to member OAuth dict using specified begin time.

**classmethod** OAuthHandler.**validate**(*oauth: str*) → dict
Deserialize and validate an OAuth string

**Raises** RequestError

OAuthHandler.**write_oauth_data**()
Dumps (treated) OAuth dict to file as JSON.

acdcli.api.oauth.**create_handler**(*path: str*)

### acdcli.api.trash module

Node trashing and restoration. https://developer.amazon.com/public/apis/experience/cloud-drive/content/trash

**class** acdcli.api.trash.**TrashMixin**
Bases: object

**list_trash**() → list
Retrieves top-level trash list

**move_to_trash**(*node_id: str*) → dict

**purge**(*node_id: str*) → dict

**restore**(*node_id: str*) → dict

### Module contents

#### ACD API

#### Usage

```
from api import client
acd_client = client.ACDClient()
root = acd_client.get_root_id()
children = acd_client.list_children(root)
for child in children:
    print(child['name'])
# ...
```

**Node JSON Format** This is the usual node JSON format for a file:

```
{
       'contentProperties': {'contentType': 'text/plain',
                             'extension': 'txt',
                             'md5': 'd41d8cd98f00b204e9800998ecf8427e',
                             'size': 0,
                             'version': 1},
       'createdBy': '<security-profile-nm>-<user>',
       'createdDate': '2015-01-01T00:00:00.00Z',
       'description': '',
       'eTagResponse': 'AbCdEfGhI01',
```

```
        'id': 'AbCdEfGhIjKlMnOpQr0123',
        'isShared': False,
        'kind': 'FILE',
        'labels': [],
        'modifiedDate': '2015-01-01T00:00:00.000Z',
        'name': 'empty.txt',
        'parents': ['0123AbCdEfGhIjKlMnOpQr'],
        'restricted': False,
        'status': 'AVAILABLE',
        'version': 1
}
```

The `modifiedDate` and `version` keys get updated each time the content or metadata is updated. `contentProperties['version']` gets updated on overwrite.

A folder's JSON looks similar, but it lacks the `contentProperties` dictionary.

`isShared` is set to `False` even when a node is actually shared.

**Caution:** ACD allows hard links for folders!

`acdcli.api.`**`new_dau`**`()`

## acdcli.bundled package

### Submodules

### acdcli.bundled.encoder module

**requests_toolbelt.multipart.encoder** This holds all of the implementation details of the MultipartEncoder

**class** `acdcli.bundled.encoder.`**`CustomBytesIO`**(*buffer=None*, *encoding='utf-8'*)
Bases: `_io.BytesIO`

**`__init__`**(*buffer=None*, *encoding='utf-8'*)

**`append`**(*bytes*)

**`len`**

**`smart_truncate`**()

**class** `acdcli.bundled.encoder.`**`FileWrapper`**(*file_object*)
Bases: `object`

**`__init__`**(*file_object*)

**`len`**

**`read`**(*length=-1*)

`acdcli.bundled.encoder.`**`IDENTITY`**(*monitor*)

**class** `acdcli.bundled.encoder.`**`MultipartEncoder`**(*fields*, *boundary=None*, *encoding='utf-8'*)
Bases: `object`

The `MultipartEncoder` object is a generic interface to the engine that will create a `multipart/form-data` body for you.

The basic usage is:

```
import requests
from requests_toolbelt import MultipartEncoder

encoder = MultipartEncoder({'field': 'value',
                            'other_field', 'other_value'})
r = requests.post('https://httpbin.org/post', data=encoder,
                  headers={'Content-Type': encoder.content_type})
```

If you do not need to take advantage of streaming the post body, you can also do:

```
r = requests.post('https://httpbin.org/post',
                  data=encoder.to_string(),
                  headers={'Content-Type': encoder.content_type})
```

If you want the encoder to use a specific order, you can use an OrderedDict or more simply, a list of tuples:

```
encoder = MultipartEncoder([('field', 'value'),
                            ('other_field', 'other_value')])
```

Changed in version 0.4.0.

You can also provide tuples as part values as you would provide them to requests' `files` parameter.

```
encoder = MultipartEncoder({
    'field': ('file_name', b'{"a": "b"}', 'application/json',
              {'X-My-Header': 'my-value'})
])
```

**Warning:** This object will end up directly in `httplib`. Currently, `httplib` has a hard-coded read size of **8192 bytes**. This means that it will loop until the file has been read and your upload could take a while. This is **not** a bug in requests. A feature is being considered for this object to allow you, the user, to specify what size should be returned on a read. If you have opinions on this, please weigh in on this issue.

**__init__** (*fields*, *boundary=None*, *encoding='utf-8'*)

**boundary_value = None**
Boundary value either passed in by the user or created

**content_type**

**encoding = None**
Encoding of the data being passed in

**fields = None**
Fields provided by the user

**finished = None**
Whether or not the encoder is finished

**len**
Length of the multipart/form-data body.

requests will first attempt to get the length of the body by calling `len(body)` and then by checking for the `len` attribute.

On 32-bit systems, the `__len__` method cannot return anything larger than an integer (in C) can hold. If the total size of the body is even slightly larger than 4GB users will see an OverflowError. This manifested itself in bug #80.

As such, we now calculate the length lazily as a property.

**parts = None**
Pre-computed parts of the upload

**read**(*size=-1*)
Read data from the streaming encoder.

**Parameters** **size** (*int*) – (optional), If provided, `read` will return exactly that many bytes. If it is not provided, it will return the remaining bytes.

**Returns** bytes

**to_string**()

**class** acdcli.bundled.encoder.**MultipartEncoderMonitor**(*encoder*, *callback=None*)
Bases: object

An object used to monitor the progress of a *MultipartEncoder*.

The *MultipartEncoder* should only be responsible for preparing and streaming the data. For anyone who wishes to monitor it, they shouldn't be using that instance to manage that as well. Using this class, they can monitor an encoder and register a callback. The callback receives the instance of the monitor.

To use this monitor, you construct your *MultipartEncoder* as you normally would.

```
from requests_toolbelt import (MultipartEncoder,
                               MultipartEncoderMonitor)
import requests

def callback(encoder, bytes_read):
    # Do something with this information
    pass

m = MultipartEncoder(fields={'field0': 'value0'})
monitor = MultipartEncoderMonitor(m, callback)
headers = {'Content-Type': montior.content_type}
r = requests.post('https://httpbin.org/post', data=monitor,
                  headers=headers)
```

Alternatively, if your use case is very simple, you can use the following pattern.

```
from requests_toolbelt import MultipartEncoderMonitor
import requests

def callback(encoder, bytes_read):
    # Do something with this information
    pass

monitor = MultipartEncoderMonitor.from_fields(
    fields={'field0': 'value0'}, callback
    )
headers = {'Content-Type': montior.content_type}
r = requests.post('https://httpbin.org/post', data=monitor,
                  headers=headers)
```

**__init__**(*encoder*, *callback=None*)

**bytes_read = None**
Number of bytes already read from the *MultipartEncoder* instance

**callback = None**
Optionally function to call after a read

**content_type**

**encoder = None**
Instance of the *MultipartEncoder* being monitored

**classmethod from_fields**(*fields*, *boundary=None*, *encoding='utf-8'*, *callback=None*)

**len = None**
Avoid the same problem in bug #80

**read**(*size=-1*)

**to_string**()

**class** acdcli.bundled.encoder.**Part**(*headers*, *body*)
Bases: object

**__init__**(*headers*, *body*)

**bytes_left_to_write**()
Determine if there are bytes left to write.

**Returns** bool – True if there are bytes left to write, otherwise False

**classmethod from_field**(*field*, *encoding*)
Create a part from a Request Field generated by urllib3.

**write_to**(*buffer*, *size*)
Write the requested amount of bytes to the buffer provided.

The number of bytes written may exceed size on the first read since we load the headers ambitiously.

**Parameters**

- **buffer** (CustomBytesIO) – buffer we want to write bytes to
- **size** (*int*) – number of bytes requested to be written to the buffer

**Returns** int – number of bytes actually written

acdcli.bundled.encoder.**coerce_data**(*data*, *encoding*)
Ensure that every object's __len__ behaves uniformly.

acdcli.bundled.encoder.**encode_with**(*string*, *encoding*)
Encoding string with encoding if necessary.

**Parameters**

- **string** (*str*) – If string is a bytes object, it will not encode it. Otherwise, this function will encode it with the provided encoding.
- **encoding** (*str*) – The encoding with which to encode string.

**Returns** encoded bytes object

acdcli.bundled.encoder.**readable_data**(*data*, *encoding*)
Coerce the data to an object with a read method.

acdcli.bundled.encoder.**reset**(*buffer*)
Keep track of the buffer's current position and write to the end.

This is a context manager meant to be used when adding data to the buffer. It eliminates the need for every function to be concerned with the position of the cursor in the buffer.

acdcli.bundled.encoder.**to_list**(*fields*)

#### acdcli.bundled.fuse module

#### Module contents

### acdcli.cache package

#### Submodules

#### acdcli.cache.cursors module

Cursor context managers

**class** acdcli.cache.cursors.**cursor**(*conn*)
Bases: object

**__init__**(*conn*)

**class** acdcli.cache.cursors.**mod_cursor**(*conn*)
Bases: object

**__init__**(*conn*)

#### acdcli.cache.db module

**exception** acdcli.cache.db.**IntegrityError**(*msg*)
Bases: Exception

**__init__**(*msg*)

**class** acdcli.cache.db.**NodeCache**(*path: str='‘*, *check=0*)
Bases: *acdcli.cache.schema.SchemaMixin*, *acdcli.cache.query.QueryMixin*, *acdcli.cache.sync.SyncMixin*, *acdcli.cache.format.FormatterMixin*

**IntegrityCheckType = {‘none’: 2, ‘quick’: 1, ‘full’: 0}**
types of SQLite integrity checks

**__init__**(*path: str='‘*, *check=0*)

**integrity_check**(*type_: {‘none’: 2, ‘quick’: 1, ‘full’: 0}*)
Performs a self-integrity check on the database.

**remove_db_file**() → bool
Removes database file.

#### acdcli.cache.format module

Formatters for query Bundle iterables. Capable of ANSI-type coloring using colors defined in LS_COLORS.

**class** acdcli.cache.format.**FormatterMixin**
Bases: object

**static id_format**(*nodes*) → ‘Generator[str]’

**long_id_format**(*nodes*) → ‘Generator[str]’

**ls_format**(*folder_id*, *folder_path=None*, *recursive=False*, *trash_only=False*, *trashed_children=False*, *long=False*, *size_bytes=False*) → ‘Generator[str]’

**path_format**(*nodes*)

**size_nlink_str**(*node*, *size_bytes=False*)
Creates a right-justified size/nlinks string.

**tree_format**(*node*, *path*, *trash=False*, *depth=0*) → 'Generator[str]'
A simple tree formatter that indicates parentship by indentation (i.e. does not display graphical branches like **tree**).

acdcli.cache.format.**color_file**(*name: str*) → str
Colorizes a file name according to its file ending.

acdcli.cache.format.**color_path**(*path: str*) → str
Colorizes a path string.

acdcli.cache.format.**color_status**(*status*)
Creates a colored one-character status abbreviation.

acdcli.cache.format.**date_str**(*time_: datetime.datetime*) → str
Creates colored date string similar to the one in ls -l.

acdcli.cache.format.**init**(*color=0*)
Disables pre-initialized coloring if never mode specified or stdout is a tty.

**Parameters** **color** – the color mode to use, defaults to auto

#### acdcli.cache.query module

**class** acdcli.cache.query.**Node**(*row*)
Bases: object

**__init__**(*row*)

**created**

**is_available**

**is_file**

**is_folder**

**is_trashed**

**modified**

**simple_name**

**class** acdcli.cache.query.**QueryMixin**
Bases: object

**calculate_usage**()

**childrens_names**(*folder_id*) → 'List[str]'

**file_size_exists**(*size*) → bool

**find_by_md5**(*md5*) → 'List[Node]'

**find_by_name**(*name: str*) → 'List[Node]'

**find_by_regex**(*regex*) → 'List[Node]'

**first_path**(*node_id: str*) → str

**get_child**(*folder_id*, *child_name*) → 'Union[Node|None]'

**get_conflicting_node**(*name: str*, *parent_id: str*)
Finds conflicting node in folder specified by *parent_id*, if one exists.

**get_folder_count**() → int

**get_node**(*id*) → 'Union[Node|None]'

**get_node_count**() → int

**get_root_node**()

**list_children**(*folder_id*, *trash=False*) → 'Tuple[List[Node], List[Node]]'

**list_trashed_children**(*folder_id*) → 'Tuple[List[Node], List[Node]]'

**num_children**(*folder_id*) → int

**num_parents**(*node_id*) → int

**resolve**(*path: str*, *trash=False*) → 'Union[Node|None]'

acdcli.cache.query.**datetime_from_string**(*dt: str*) → datetime.datetime

### acdcli.cache.schema module

**class** acdcli.cache.schema.**SchemaMixin**
Bases: object

**create_tables**()

**drop_all**()

**init**()

### acdcli.cache.sync module

Syncs Amazon Node API objects with SQLite database.

**class** acdcli.cache.sync.**SyncMixin**
Bases: object

Sync mixin to the *NodeCache*

**insert_files**(*files: list*)

**insert_folders**(*folders: list*)
Inserts list of folders into cache. Sets 'update' column to current date.

**Parameters** **folders** – list of raw dict-type folders

**insert_node**(*node: dict*)
Inserts single file or folder into cache.

**insert_nodes**(*nodes: list*, *partial=True*)
Inserts mixed list of files and folders into cache.

**insert_parentage**(*nodes: list*, *partial=True*)

**remove_purged**(*purged: list*)
Removes purged nodes from database

**Parameters** **purged** – list of purged node IDs

acdcli.cache.sync.**gen_slice**(*list_*, *length=100*)

acdcli.cache.sync.**placeholders**(*args*)

#### Module contents

### acdcli.plugins package

#### Submodules

#### acdcli.plugins.template module

This is a template that you can use for adding custom plugins.

**class** acdcli.plugins.template.**TestPlugin**
Bases: *acdcli.plugins.Plugin*

**MIN_VERSION = '0.3.1'**

**classmethod action**(*args: argparse.Namespace*) → int
This is where the magic happens. Return a zero for success, a non-zero int for failure.

**classmethod attach**(*subparsers: argparse.ArgumentParser*, *log: list*, ***kwargs*)
Attaches this plugin to the top-level argparse subparser group :param subparsers the action subparser group :param log a list to put initialization log messages in

**registry = {<class 'acdcli.plugins.template.TestPlugin'>}**

#### Module contents

**class** acdcli.plugins.**Plugin**
Bases: object

Plugin base class. May be subject to changes.

**MAX_VERSION = None**

**MIN_VERSION = None**

**static action**(*args: argparse.Namespace*)

**classmethod attach**(*subparsers: argparse.ArgumentParser*, *log: list*, ***kwargs*)

**classmethod check_version**(*version: str*) → bool

**registry = {<class 'acdcli.plugins.template.TestPlugin'>}**

**class** acdcli.plugins.**RegisterLeafClasses**(*name*, *bases*, *nmspc*)
Bases: type

**__init__**(*name*, *bases*, *nmspc*)

## acdcli.utils package

### Submodules

#### acdcli.utils.hashing module

**class** acdcli.utils.hashing.**IncrementalHasher**
Bases: object

**__init__**()

**get_result**() → str

**hasher**

**update**(*chunk*)

acdcli.utils.hashing.**hash_file**(*file_name: str*) → str

acdcli.utils.hashing.**hash_file_obj**(*fo*) → str

#### acdcli.utils.progress module

**class** acdcli.utils.progress.**FileProgress**(*total_sz: int*, *current: int=0*)
Bases: object

**__init__**(*total_sz: int*, *current: int=0*)

**current**

**done**()

**reset**()

**status**

**total**

**update**(*chunk*)

**class** acdcli.utils.progress.**MultiProgress**
Bases: object

Container that accumulates multiple FileProgress objects

**__init__**()

**add**(*progress: acdcli.utils.progress.FileProgress*)

**end**()

**print_progress**()

acdcli.utils.progress.**file_size_str**(*num: int*, *suffix='B'*) → str

acdcli.utils.progress.**speed_str**(*num: int*, *suffix='B'*, *time_suffix='/s'*) → str

acdcli.utils.progress.**time_str**(*num: float*) → str

#### acdcli.utils.threading module

**class** acdcli.utils.threading.**QueuedLoader**(*workers=1*, *print_progress=True*, *max_retries=0*)
Bases: object

Multi-threaded loader intended for file transfer jobs.

**MAX_NUM_WORKERS = 8**

**MAX_RETRIES = 4**

**REFRESH_PROGRESS_INT = 0.3**

**__init__**(*workers=1*, *print_progress=True*, *max_retries=0*)

**add_jobs**(*jobs: list*)

**Parameters** **jobs** – list of partials that return a RetryRetVal and have a pg_handler kwarg

**start**() → int
Starts worker threads and, if applicable, progress printer thread. :returns: accumulated return value

#### acdcli.utils.time module

acdcli.utils.time.**datetime_to_timestamp**(*dt: datetime.datetime*) → float

#### Module contents

### 11.1.2 Submodules

### 11.1.3 acdcli.acd_fuse module

### 11.1.4 Module contents

## 11.2 acdcli

## 11.3 TODO

### 11.3.1 General / API

- switch to multiprocessing (?)
- metalink support (?)

### 11.3.2 API

- support of node labels
- support for assets (?)
- favorite support (feature not yet announced officially)
- rip out the Appspot authentication handler
- fix upload of 0-byte streams

### 11.3.3 CLI

- unify the find action
- check symlink behavior for different Python versions (#95)

### 11.3.4 FUSE

- invalidate chunks of StreamedResponseCache (implement a time-out)
- respect flags when opening files
- use a filesystem test suite

### 11.3.5 File Transfer

- more sophisticated progress handler that supports offsets
- copy local mtime on upload (#58)
- add path exclusion by argument for download

### 11.3.6 User experience

- shell completion for remote directories (#127)
- even nicer help formatting
- log coloring

### 11.3.7 Tests

- cache methods
- more functional tests
- fuse module

### 11.3.8 Documentation

- write how-to on packaging plugins (sample setup.py)

CHAPTER 12

# Overview

**acd_cli** provides a command line interface to Amazon Cloud Drive and allows mounting your cloud drive using FUSE for read and write access. It is currently in beta stage.

CHAPTER 13

# Node Cache Features

- caching of local node metadata in an SQLite database
- addressing of remote nodes via a pathname (e.g. `/Photos/kitten.jpg`)
- file search

CHAPTER 14

# CLI Features

- tree or flat listing of files and folders
- simultaneous uploads/downloads, retry on error
- basic plugin support

## 14.1 File Operations

- upload/download of single files and directories
- streamed upload/download
- folder creation
- trashing/restoring
- moving/renaming nodes

CHAPTER 15

# Documentation

The full documentation is available at https://acd-cli.readthedocs.org.

CHAPTER 16

# Quick Start

Have a look at the *known issues*, then follow the setup guide and authorize. You may then use the program as described in the usage guide.

CHAPTER 17

# CLI Usage Example

In this example, a two-level folder hierarchy is created in an empty cloud drive. Then, a relative local path `local/spam` is uploaded recursively using two connections.

```
$ acd_cli sync
  Syncing...
  Done.

$ acd_cli ls /
  [PHwiEv53QOKoGFGqYNl8pw] [A] /

$ acd_cli mkdir /egg/
$ acd_cli mkdir /egg/bacon/

$ acd_cli upload -x 2 local/spam/ /egg/bacon/
  [##################################]   100.0% of  100MiB  12/12  654.4KB/s

$ acd_cli tree
  /
      egg/
          bacon/
              spam/
                  sausage
                  spam
  [...]
```

The standard node listing format includes the node ID, the first letter of its status and its full path. Possible statuses are “AVAILABLE” and “TRASH”.

CHAPTER 18

# Known Issues

It is not possible to upload files using Python 3.2.3, 3.3.0 and 3.3.1 due to a bug in the http.client module.

## 18.1 API Restrictions

- the current upload file size limit is 50GiB
- uploads of large files >10 GiB may be successful, yet a timeout error is displayed (please check the upload by syncing manually)
- storage of node names is case-preserving, but not case-sensitive (this should not concern Apple users)
- it is not possible to share or delete files

CHAPTER 19

# Contribute

Have a look at the contributing guidelines.

CHAPTER 20

# Recent Changes

## 20.1 0.3.1

- general improvements for FUSE
- FUSE write support added
- added automatic logging
- sphinx documentation added

## 20.2 0.3.0

- FUSE read support added

## 20.3 0.2.2

- sync speed-up
- node listing format changed
- optional node listing coloring added (for Linux or via LS_COLORS)
- re-added possibility for local OAuth

## 20.4 0.2.1

- curl dependency removed
- added job queue, simultaneous transfers
- retry on error

## 20.5 0.2.0

- setuptools support

- workaround for download of files larger than 10 GiB
- automatic resuming of downloads

## a

## Symbols



## A

## B



## C



## D



## E

## F



## G



## H



## I

## L



## M



## N



## O



## P



## Q

## R



## S



## T



## U



## V



## W