ArtifactDB Command Line Interface (CLI)

Tutorial, design and command reference

Sébastien Lelong / DSSC / Genentech

2023-08-14

Contents

Acknowledgments

The implementation of this ArtifactDB Command Line Interface is heavily based on the python-based generic client, artifactdb-client, developed and maintained by Max Hargreaves. It provides low level access to common ArtifactDB REST API endpoints, authentication and all the “nitty-gritty” details required to properly talk to such instances, the proper way.

Marek Brys proof-read this tutorial (thanks for the feedback) and contributed to the custom CA certificates installation, you now know who to ask if still having issues :)

Introduction

The ArtifactDB framework originated within the Genomics Platform, here at Genentech. Main use cases were about supporting data and metadata management of genomics datasets, in a cloud-based approach. ArtifactDB clients have been developed with the purpose of providing bioinformaticians and computional biologists a convenient way to access these datasets, with the tools they are the most familiar with: R session, RStudio, Jupyter notebooks, etc… that is, analytical environments, close to scientific use cases.

These clients provide not only access to the data and metadata by interfacing with ArtifactDB REST API endpoints on the backend side, but also manage the data model and its representation in the context of these analytical environments. For instance, `dsassembly` handles the conversion between a Multi Assay Experiment (MAE), an in-memory R structure, and its "deconstructed" equivalent found in an ArtifactDB instance.

In the process of building these rich, heavy and sophisticated clients, a generic use case has fallen between the cracks: accessing an ArtifactDB instance in the most simplest and direct approach, from our beloved terminal. The ArtifactDB Command Line Interface (CLI) goal is to bridge that gap[1], by providing an easy way to connect to any ArtifactDB instances and perform common operations such as searching metadata, uploading and downloading projects, managing permissions, etc…

Who should use ArtifactDB CLI? Anyone who needs to access an ArtifactDB instance at a low level, ie. not necessarily from an analytical environments where these heavy clients mentioned above are more suitable. The CLI is a convenient way to prototype, interface with an instance, on a file by file basis. Storing and managing individual artifacts is a generic and very simple use case, yet very useful, and possibly constituting a first step in maturing an instance later with a more advanced, specialized client.

[1] The presence of Iris, goddess of the rainbow, found on the cover of this document, is not a random choice. Iris is also responsible for carrying the water of the River Styx to Olympus (aka the Self Service component of the ArtifactDB platform). The Greek mythology mentions the water would render unconscious for one year any god or goddess who lied. I'm still not sure what this could mean in this context…

Installation

The `artifact-cli` package can be installed from PyPi:

```
pip install 'artifactdb-cli'
```

After the installation, an executable named `adb` is available. We can verify that by typing `adb version`. Make sure you have at least version `0.3.0`.

```
$ adb version
```

gives something similar to:

```
ArtifactDB-CLI version '0.3.0'
ArtifactDB-client version '0.3.2'
```

The source code found in the Git repo https://github.com/artifactdb/artifactdb-cli can also be used:

```
$ git clone git@ssh.github.com:artifactdb/artifactdb-cli.git
...
$ cd artifactdb-cli
$ pip install .
```

It is recommended to use a python virtual environment or Conda to isolate the installation in a separate location.

Tutorial

Obtaining help

With the executable `adb` installed, we can start to explore what we can do with it. Generally speaking, `adb` can take a command, possibly sub-command as arguments, following by command-specific options. This is revealed in the help message:

```
$ adb --help

 Usage: adb [OPTIONS] COMMAND [ARGS]...

   Options
  --install-completion          Install completion for the current shell.
  --show-completion             Show completion for the current shell, to copy it or
      customize the installation.
  --help                        Show this message and exit.

Commands
  context             Manage ArtifactDB contexts (connections, clients, ...)
  download            Download artifacts.
  job                 Manage jobs (indexing, ...)
  login               Switch to authenticated access. Only effective within an ArtifactDB
      shell.
  logout              Switch to anonymous access. Only effective within ArtifactDB shell.
  permissions         Manage project's permissions
  plugins             Manage CLI plugins
  search              Searching metadata documents, using active context.
  shell               Launch interactive shell
  upload              Upload artifacts.
  version             Print the CLI version information
```

This displays the main commonds such as `context`, `download`, `job`, etc... The design of these commands is inspired by the AWS CLI, with a grain of salt coming from `kubectl`. To obtain help on a specific commands, we can add `--help` to one of the main command, eg. `adb context --help` :

```
$ adb context --help
Usage: adb context [OPTIONS] COMMAND [ARGS]...

  Manage ArtifactDB contexts (connections, clients, ...)

Options:
  --help  Show this message and exit.

Commands:
  create  Create a new ArtifactDB context with connection details.
```

```
  list    List available ArtifactDB contexts
  show    Show ArtifactDB context details
  use     Set given context as current one
```

This `context` command reveals a set of sub-commands: `create`, `list`, `show`, and `use`. Proceeding further, `adb context create --help` display help messages for that `create` sub-command:

```
$ adb context create --help
Usage: adb context create [OPTIONS] URL

  Create a new ArtifactDB context with connection details.

Arguments:
  URL  URL pointing to the REST API root endpoint  [required]

Options:
  --auth-url TEXT                 Keycloak auth URL (contains realm name, eg.
                                  `awesome` https://mykeycloak.mycompany.com/r
                                  ealms/awesome)

  --name TEXT                     Context name. The instance name (if exposed)
                                  is used by default to name the context

  --auth-client-id TEXT           Client ID used for authentication. The
                                  instance's main client ID (if exposed) is
                                  used by default

  --auth-username TEXT            Username used in authentication, default to
                                  `whoami`

  --project-prefix TEXT           Project prefix used in that context. If the
                                  instance exposes that information, a
                                  selection can be made, otherwise, the
                                  instance's default is used

  --auth-service-account-id TEXT  Create a context for a service account,
                                  instead of current user

  --force / --no-force            Don't ask for confirmation before creating
                                  the context  [default: False]

  --help                          Show this message and exit.
```

We now know the theory of fishing, it's time to practice.

Setting up auto-completion

There are a lot of commands, sub-commands, arguments, even more choices as we'll set up contexts. `adb` provides a convenient way to install auto-completion and enable abusive usage of the TAB key to complete what we're looking for.

```
$ adb --install-completion
bash completion installed in /data/home/lelongs/.bash_completions/adb.sh
Completion will take effect once you restart the terminal
```

This instruction tries to detect the current shell, here Bash, and generate the corresponding completion code file. From there, we need to restart the terminal to load this code.

Keep in mind the CLI is implemented in python, which is not the faster programming language available on Earth. Auto-completion may take unusual extra milliseconds, even a second on heavy loaded system such as tortured login node of an HPC. This latency shouldn't be seen as a source of frustation and despair, but rather as an invitation to reflect on life in general, and the meaning of time specifically, and its relation to suffering.

Using the shell

Autocompletion is great, and though this should be seen as an invitation to reflect on our lives, it can be slow and annoying. Entering the "shell":

```
$ adb shell
Welcome to the ArtifactDB shell, type `help` for available commands
No existing configuration file found at '/data/home/lelongs/.config/artifactdb-cli/config',
    creating one
No active context found.
To list available contexts, use context list
To point to an existing one with context use
To create a new one with context create
adb>
adb> help

Documented commands (type help <topic>):
========================================
context  download  job  permissions  plugins  search  shell  upload

Undocumented commands:
======================
exit  help  quit

adb>
```

The command `adb shell` loads everything in memory, and give access to an ArtifactDB shell where we can type the same commands as if we were typing them from our Bash terminal, without the `adb` part. And autocomplete is now super fast, try it! The shell itself also maintains a history file, so we can nagivate back to previous commands we typed using the up/down arrow keys. Yes, we're happy, we defeated time and are now free from suffering.

Configuration files

When we access the ArtifactDB shell for the first time, `adb` found no previous configuration file, and created one. The location is supposed to be a standard one, and depends on which system you're using the CLI. Typically, `adb` will use `$HOME/.config/artifactdb-cli` folder on Linux systems,

and "$HOME/Library/Application Support/artifactdb-cli" on MacOS. Several files will be created there: to store contexts (as we'll see next), registered plugins, search profiles, etc… This folder can be copied to the right location on another system, to reuse these elements.

Managing contexts

So far, we've done nothing about interacting with an ArtifactDB instance. Let's do that now, by creating a context. A context defines which ArtifactDB instance we want to talk to, and how we want to interact with it[1]. In the context of this tutorial, we'll be using a demo instance named `olympusapi2`, hosted by Olympus, the Self-Service component of the ArtifactDB platform. Please keep in mind this instance is used to demo purpose, is shared with at least anyone who's following that tutorial. No important and/or confidential data should be put there. And data can be wiped at any time. This demo instance can be reached at https://democli.api.artifactdb.io/v1, the root endpoint, and this is exactly what we'll need and provide to the CLI, when creating a context:

```
$ adb context create https://democli.api.artifactdb.io/v1
What is the name of the context to create?  (olympus-api-2-uat):
Found auth issuer URL https://todo
Select the client ID that will be used for authentication [olympus-client2] (olympus-client2)
    :
What is the username used during authentication (lelongs):
Select a project prefix:  [PRJ/test-PRJ] (PRJ):
Create new context:
auth:
  client_id: olympus-client2
  service_account_id: null
  url: https://todo
  username: lelongs
name: olympus-api-2-uat
project_prefix: PRJ
url: https://democli.api.artifactdb.io/v1

Confirm creation? [y/n]: y
Authenticating user: 'lelongs'.
Authenticating user: 'lelongs'.
Password:
Successfully authenticated.
```

We need to answer some questions, but `adb` did a great job providing default yet meaningful values. How is that magic possible? `adb` connects to this URL and tries to discover as much information as possible: the name of the instance, its environment, the authentication issuer URL, the available project prefixes and their usage, etc… Not all ArtifactDB instances are able to provide this information, only most recent ones. If `adb` cannot guess a value, it will ask you to answer. Specifically, it may complain about not finding the authentication issuer URL, which can be tricky to guess. As a rule of thumb:

[1] If you're familiar to Kubernetes and `kubectl` context, the idea is the same.

- if creating a context pointing to a production instance, use `--auth-url=https://todo`
- if dev or uat instance, use `--auth-url=https://todo`

Because `olympusapi2` is recent, it provides that information, and `adb` works for you.

The same creation procedure can be achieved from `adb shell`. Note we don't need to type `adb` since we're already in the CLI shell:

```
$ adb shell
adb> context create https://democli.api.artifactdb.io/v1
...
```

Note: You may find that questions are asked one after the other without returning to the beginning of the line. It happens on some systems, depending on the terminal configuration. It's not elegant, but still works… (yes, this can be seen as a bug)

Once the context is created, we can use it:

```
adb> context list
['olympus-api-2-uat']
```

```
adb> context use olympus-api-2-uat
Switched to context 'olympus-api-2-uat': https://democli.api.artifactdb.io/v1
```

A default context is now active. The next time we use `adb`, it will use that context by default. In order to know which one is used, `adb shell` reports that at the beginning, but we can also use:

```
adb> context show
auth:
  client_id: olympus-client2
  service_account_id: null
  url: https://todo
  username: lelongs
name: olympus-api-2-uat
project_prefix: PRJ
url: https://democli.api.artifactdb.io/v1
```

`adb context show`, without a context name, shows the default, active one. We can add a context name to reveal the configuration values of a specific context, this does not change the default context, only `adb context use ...` can do that.

Uploading artifacts

We're now ready to interact with that demo instance. Let's upload some data! With the demo instance we're using, we can upload any sort of data files. We'll create a folder with some text files, and upload the whole:

```
$ mkdir /tmp/staging_dir
$ echo "I love ArtifactDB" > /tmp/staging_dir/file1.txt
```

```
$ echo "and I love the CLI even more" > /tmp/staging_dir/file2.txt
```

We'll also create minimal metadata files. This step depends on the models that are registered in the instance. In our case, the demo instance only requires the field `path` to be defined in a JSON file. Let's create these metadata files now[2]:

```
$ echo '{"path":"file1.txt"}' > /tmp/staging_dir/file1.txt.json
$ echo '{"path":"file2.txt"}' > /tmp/staging_dir/file2.txt.json
```

Uploading files is done using the command `upload`. There are plenty of arguments this command can take, we'll just explore a fraction there, but `adb upload --help` should provide all the details. For now:

```
$ adb shell
Welcome to the ArtifactDB shell, type `help` for available commands
Active context 'olympus-api-2-uat': https://democli.api.artifactdb.io/v1
```

```
adb> upload /tmp/staging_dir
Authenticating user: 'lelongs'.
Successfully authenticated.
23:13:37 -> Collating project.
23:13:37 -> Validating JSON metadata.
23:13:37 -> JSONs validated.
23:13:37 -> Project collated, attempting upload.
100%|.....................................| 4/4 [00:00<00:00,  4.12file/s]
Clearing upload info.
23:13:43 -> Upload completed.
Overwriting existing context 'olympus-api-2-uat'
Job created for project PRJ000000021@1:
SubmittedJobInfo(
    job_id='8433906d-d087-4a27-81c0-6ba74e60bcc9',
    job_url='http://democli.api.artifactdb.io/v1/jobs/8433906d-d087-4a27-81c0-6ba74e60bcc9',
    path='/jobs/8433906d-d087-4a27-81c0-6ba74e60bcc9',
    status='accepted'
)
```

And that's it, our two text files were uploaded. There are several information returned there:

- We were assigned a new project ID and version, `PRJ000000021@1`, meaning project ID `PRJ000000021` and version 1. If you follow this tutorial, you will probably get a different project ID.
- A job was created, with an ID and a status "accepted". This is an asynchronous job created by the instance to index the metadata. When the job is done, the status becomes `success` or `failure`. We'll see in the next section how to check these job statuses.

There are other interesting options. Adding `--confirm` and `--verbose` will display interesting information, such as the number of files to upload, the upload mode, the permissions which will be

[2] Some instances don't even need to have metadata files uploaded, some inspectors, when declared on the backend side, can automatically create these JSON files for us. At the time of this writing though, the demo instance has no such inspectors running, so we need to provide the metadata files. The model is an absolute minimal one though, we just need to provide that field "path", that's all.

applied at upload time, etc…

```
adb> upload /tmp/staging_dir --confirm --verbose
Authenticating user: 'lelongs'.
Successfully authenticated.
Summary
Uploading 4 files from folder /tmp/staging_dir
As a new project
Using presigned upload mode
Setting following permissions:
owners:
- lelongs
read_access: viewers
scope: project
write_access: owners

Proceed? [y/n] (n):
```

When enabled on the instance, we can use another upload method than the default one (which is using S3 presigned URLs). If uploading files bigger than 5GiB, presigned URLs can't be used, and even when files are bigger than 100MiB, it's recommanded to use an upload method based STS credentials, such as `sts:boto3`.

```
adb> upload /tmp/staging_dir --confirm --verbose --upload-mode sts:boto3
Authenticating user: 'lelongs'.
Successfully authenticated.
Summary
Uploading 4 files from folder /tmp/staging_dir
As a new project
Using sts:boto3 upload mode
Setting following permissions:
owners:
- lelongs
read_access: viewers
scope: project
write_access: owners

Proceed? [y/n] (n):
```

Note if you proceed further, you will get an error, mentioning "Unable to obtain STS credentials, not supported". This demo instance doesn't support STS credentials at this moment…

By default, as revealed by the `--verbose` option, permissions default to "private": read access limited to a list of explicit viewers, read/write access to owners, one of the owners being the person uploading the files. Several options can be used to adjust the permissions at upload time, such as `--owners`, `--viewers`, `--read-access`, `--write-access`, and `--permissions-json` (which allows to fully declare a permissions profile). For instance, if we want to add "anotherperson" as an owner, and make the project public, we can specify:

```
adb> upload /tmp/staging_dir --confirm --verbose --owners lelongs,anotherperson --read-access
     public
Authenticating user: 'lelongs'.
Successfully authenticated.
Summary
Uploading 4 files from folder /tmp/staging_dir
As a new project
```

```
Using presigned upload mode
Setting following permissions:
owners:
- lelongs
- anotherperson
read_access: public
scope: project
write_access: owners

Proceed? [y/n] (n):
```

Permissions can also be adjusted after the creation of a project. We'll address this use case in a later section.

One last example is to add a new version to an existing project, by specifying the project ID with the option `--project-id`. Let's do this, but first, for instructional purpose, we will create an incorrect metadata file, not containing the mandatory field "path". Moving back to the terminal (remember to replace `PRJ000000021` with the project ID you were assigned ealier):

```
adb> exit
$ echo '{"chemin": "fichier3.txt"}' > /tmp/staging_dir/file3.txt.json
```

```
$ adb upload /tmp/staging_dir --project-id PRJ000000021 --confirm --verbose
Authenticating user: 'lelongs'.
Successfully authenticated.
Summary
Uploading 5 files from folder /tmp/staging_dir
As a new version within project PRJ000000021
Using presigned upload mode
Setting following permissions:
owners:
- lelongs
read_access: viewers
scope: project
write_access: owners

Proceed? [y/n] (n): y
23:52:02 -> Collating project.
23:52:02 -> Validating JSON metadata.
23:52:02 -> JSONs validated.
23:52:02 -> Project collated, attempting upload.
100%|.....................................| 5/5 [00:00<00:00,  6.38file/s]
Clearing upload info.
23:52:09 -> Upload completed.
Overwriting existing context 'olympus-api-2-uat'
Job created for project PRJ000000021@2:
SubmittedJobInfo(
    job_id='b0b1983f-8d6a-43c6-94f2-0fd00d9201fb',
    job_url='http://democli.api.artifactdb.io/v1/jobs/b0b1983f-8d6a-43c6-94f2-0fd00d9201fb',
    path='/jobs/b0b1983f-8d6a-43c6-94f2-0fd00d9201fb',
    status='accepted'
)
```

Another indexing job was created, status "accepted". We can also see a new version 2 was created within our project `PRJ000000021`, as shown in the line `Job created for project PRJ000000021@2`.

Monitoring jobs

We uploaded some data files, got a job ID in return, how can we check if that indexing job worked or not? We can use the command `job` for that purpose:

```
$ adb shell
adb> job check
Authenticating user: 'lelongs'.
Successfully authenticated.
Job '8433906d-d087-4a27-81c0-6ba74e60bcc9'
 Success
 indexed_files: 2
 project_id: PRJ000000021
 version: 1

Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
 Failure
 null
 ...
```

When using the command `job check`, the CLI looks for jobs which were registered on the configuration files. When we uploaded our files, the CLI automatically captured the job details so we don't have to remember the job identifiers for instance.

The first job succeed, two files were indexed[3], but the second failed, as expected since we provided incorrect metadata on purpose. If we try to check the job status again:

```
adb> job check
Authenticating user: 'lelongs'.
Successfully authenticated.
Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
 Failure
 null
 ...
```

This time, the job which succeeded is not listed anymore, but the one which failed is still there. Indeed, the CLI keeps some job details depending on their statuses. Knowning the first job succeeded is nice, but we honestly don't need that information anymore, so the CLI "pruned" that job. Failure is painful and needs investigation, `--verbose` to the rescue:

```
adb> job check --verbose
Authenticating user: 'lelongs'.
Successfully authenticated.
Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
Failure
...
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/site-packages/celery/app/trace.py", line 385, in trace_task
    R = retval = fun(*args, **kwargs)
  File "/usr/local/lib/python3.8/site-packages/celery/app/trace.py", line 650, in
      __protected_call__
    return self.run(*args, **kwargs)
```

[3] But we uploaded four files, why only two files were indexed? Because only the JSON metadata files are indexed. We had two data files, and two metadata files, one for each, so two files indexed in the end.

```
  File "/usr/local/lib/python3.8/site-packages/celery/app/base.py", line 487, in run
    return task._orig_run(*args, **kwargs)
  File "/usr/local/lib/python3.8/site-packages/artifactdb/backend/tasks/core.py", line 35, in
      index
    num = self._app.manager.index_project(project_id,*args,**kwargs)
  File "/usr/local/lib/python3.8/site-packages/artifactdb/backend/managers/common.py", line
      68, in index_project
    return self.do_index_project(project_id=project_id,version=version,
  File "/usr/local/lib/python3.8/site-packages/artifactdb/backend/managers/common.py", line
      158, in do_index_project
    docs = self.get_documents(project_id,version)
  File "/usr/local/lib/python3.8/site-packages/artifactdb/backend/managers/common.py", line
      358, in get_documents
    doc = self.get_document(key,links)
  File "/usr/local/lib/python3.8/site-packages/artifactdb/backend/managers/common.py", line
      409, in get_document
    assert path, "Missing 'path' or 'PATH' field"
AssertionError: Missing 'path' or 'PATH' field
```

We get a “nice” traceback with, at the end, an explanation: `Missing 'path' or 'PATH' field`. Not a surprise.

So, will that job in status failure always be there? We know it failed, we know why, how to get rid of it? We can tell the CLI which job to prune to their status. Here, we want to remove a job in status “failure”, so we’ll use the option `--prune failure`.

```
adb> job check --prune failure
Authenticating user: 'lelongs'.
Successfully authenticated.
Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
Failure
null
...
```

No more jobs are registered, a subsequent call will let us know about that:

```
adb> job check
No jobs recorded in current context, nothing to check
```

Finally, what if we know the job ID but this job was not registered automatically? By default, `adb job check` only look for registered jobs, but we can still specify an ID (here, we query the job in failure again

```
adb> job check b0b1983f-8d6a-43c6-94f2-0fd00d9201fb
Authenticating user: 'lelongs'.
Successfully authenticated.
Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
 Failure
 null
 ...
```

Specifying a job ID not only (tries to) retrieve the job details, but also auto-register it. So our failing job is back, stored again in our configuration! Let’s remove it again, with `--prune all`, which is a more drastic approach as it removes all registered jobs, no matter what their statuses are.

```
adb> job check --prune all
```

```
Authenticating user: 'lelongs'.
Successfully authenticated.
Job 'b0b1983f-8d6a-43c6-94f2-0fd00d9201fb'
 Failure
 null
 ...
```

```
adb> job check
No jobs recorded in current context, nothing to check
```

Searching metadata

Now that we have our new project uploaded and indexed, we can search for it, with a command conveniently named `search`. By default, if nothing more is specified, this command performs a wildcard search on the whole instance. If lots of results are returned, a pagination is put in place, asking if we want to get more results. For demo purpose, we'll restrict our search to our project, `PRJ000000021`:

```
adb> search PRJ000000021
Authenticating user: 'lelongs'.
Successfully authenticated.
_extra:
  file_size: 18
  gprn: gprn:uat:olympusapi2::artifact:PRJ000000021:file1.txt@1
  id: PRJ000000021:file1.txt@1
  index_name: olympusapi2-uat-myschema-20230119001122
  location:
    s3:
      bucket: gred-olympus-dev-olympusapi2-v1-uat
    type: s3
  meta_indexed: '2023-02-06T23:37:49.117875+00:00'
  meta_uploaded: '2023-02-06T23:37:46+00:00'
  metapath: file1.txt.json
  numerical_revision: 1
  permissions:
    owners:
    - lelongs
    read_access: viewers
    scope: project
    write_access: owners
  project_id: PRJ000000021
  revision: NUM-1
  transient: null
  uploaded: '2023-02-06T23:37:46+00:00'
  version: '1'
path: file1.txt

---
_extra:
  file_size: 29
  gprn: gprn:uat:olympusapi2::artifact:PRJ000000021:file2.txt@1
  id: PRJ000000021:file2.txt@1
  index_name: olympusapi2-uat-myschema-20230119001122
  location:
    s3:
      bucket: gred-olympus-dev-olympusapi2-v1-uat
```

```
    type: s3
  meta_indexed: '2023-02-06T23:37:49.118175+00:00'
  meta_uploaded: '2023-02-06T23:37:46+00:00'
  metapath: file2.txt.json
  numerical_revision: 1
  permissions:
    owners:
    - lelongs
    read_access: viewers
    scope: project
    write_access: owners
  project_id: PRJ000000021
  revision: NUM-1
  transient: null
  uploaded: '2023-02-06T23:37:46+00:00'
  version: '1'
path: file2.txt

---
No more results
```

We can find our two indexed files back[4]. A lot of other metadata fields are also returned, under the key "_extra": this is automatically added by all ArtifactDB instance, to provide context for the entry, such as the data location, permissions, version, etc… We can narrow down the fields returned in the search results, with the `--fields` option. Let's say we're only interested in the ArtfactDB ID, and the file size:

```
adb> search PRJ000000021 --fields=_extra.id,_extra.file_size
Authenticating user: 'lelongs'.
Successfully authenticated.
_extra:
  file_size: 18
  id: PRJ000000021:file1.txt@1

---
_extra:
  file_size: 29
  id: PRJ000000021:file2.txt@1

---
No more results
```

This is better. We spent a lot of time designing these search parameters, and we would be a shame if we had to think about these again. Luckily, we can store these parameters as a "search profile", with the `--save` option, and use that profile later with the `--load` option. We can override any profile parameters, and even load and save the profile at the same time, to adjust the profile content:

```
adb> search PRJ000000021 --fields=_extra.id,_extra.file_size --save=my_search_profile
...
```

```
adb> search --load=my_search_profile
Authenticating user: 'lelongs'.
Successfully authenticated.
_extra:
```

[4] All fields are search by default, so search for `PRJ000000021` is enough, but we could have been more explicit, `adb> search _extra.project_id:PRJ000000021` to search the actual field storing the project ID.

```
  file_size: 18
  id: PRJ000000021:file1.txt@1

---
_extra:
  file_size: 29
  id: PRJ000000021:file2.txt@1
---
No more results

adb> search --load=my_search_profile --fields=path,_extra.id --save=my_search_profile
Authenticating user: 'lelongs'.
Successfully authenticated.
_extra:
  id: PRJ000000021:file1.txt@1
path: file1.txt

---
_extra:
  id: PRJ000000021:file2.txt@1
path: file2.txt

---
No more results
adb> search --load=my_search_profile
Authenticating user: 'lelongs'.
Successfully authenticated.
_extra:
  id: PRJ000000021:file1.txt@1
path: file1.txt

---
_extra:
  id: PRJ000000021:file2.txt@1
path: file2.txt

---
No more results
```

Search profiles can be managed using `--ls`, `--delete`, and `--show`, please refer to the help section for more.

Searching using boolean operations is also possible, the first search argument must the search query, so we'll have to use double quotes for that:

```
adb> search "PRJ000000021 AND _extra.file_size:29" --fields=path
Authenticating user: 'lelongs'.
Successfully authenticated.
path: file2.txt

---
No more results
```

Downloading artifacts

The command `download` can be used to retrieve the data files back on our local computer. We can download either one single artifact, or the data files within a specific project and version. The

destination folder is created if it doesn't exist, but by default files are downloaded from the location where the `adb` command run. Let's illustrate this, by downloading the files in version 1. We use the notation `{project_id}@{version}`:

```
adb> download PRJ000000021@1
project_id: PRJ000000021
version: 1
path: None
Authenticating user: 'lelongs'.
Successfully authenticated.
PRJ000000021:file1.txt@1: 100%|.....................................| 18.0/18.0
    [00:00<00:00, 31.1kB/s]
PRJ000000021:file2.txt@1: 100%|.....................................| 29.0/29.0
    [00:00<00:00, 51.4kB/s]
```

To download a single artifact, we need to specify an ArtifactDB ID, `{project_id}:{path}@{version}`:

```
adb> download PRJ000000021:file1.txt@1 /tmp/my_dest_folder
```

By default, the CLI will refuse to overwrite existing files on the local computer, unless `--overwrite` is explicitly passed.

Note: The download mecanism is currently using S3 presigned URLs, but an upcoming improvement will allow to download using STS credentials (when enabled on the instance's side), just like the upload mode seen ealier.

Managing permissions

We uploaded some artifacts in our new project `PRJ000000021`. The world is already a better place, but we can do more: share our project with everybody! Enter the command `permissions`. In ArtifactDB, permissions are fine-grained, and can sometimes be tricky to manage, but the CLI provides tools to make sure we're doing the right thing when changing permissions, specifically when using the option `--verbose`.

There are many different ways to change permissions on an existing projects. In order to better understand how, let's review what a permissions profile is:

- Permissions can be defined at different scopes, with the `scope` field: `version`, `project` and `global`. When permissions are defined within a specific version, all artifacts within that versions inherits from these permissions. If no version-level permissions, project-level permissions are considered (this is the most common case), and if still no permissions found, global-level permissions are considered (very rare case). If still no global permissions, the project is left without any permissions declared (unless the instance doesn't allow that) or default permissions

are set by the instance itself (if that instance has default permissions put in place). This is even more rare, I would even say unexpected…

- We can declare who can read the artifacts, using the field `read_access`, and read/write artifacts, using the field `write_access`. Allowed values are the same between `read_access` and `write_access`, only the access mode is different (read-only vs. read/write):
  - `public`: anyone can be access the data, even anonymous
  - `authenticated`: users need at least to be authenticated but there's no specific checks after that.
  - `viewers` or `owners`: only users declared in the "viewers" or "owners" lists can access the artifacts
  - `none`: access is disabled
- `viewers` and `owners` are list of users, defining two distinct populations that can be used in `read_access` and `write_access`. By convention, viewers are used in `read_access` and `owners` in `write_access`. An owner has at least the same permissions as a viewer (if an owner can write data, she can also read it back).

Let's retrieve current permissions:

```
adb> permissions show PRJ000000021@1
Authenticating user: 'lelongs'.
Successfully authenticated.
owners:
- lelongs
read_access: viewers
scope: project
viewers: null
write_access: owners
```

We specified the version 1, with `PRJ000000021@1`, but we can see that the permissions are actually found at the project-level, `scope: project`. There is no permissions specific to the version, all artifacts within that version have the same permissions declared for project `PRJ000000021`. There's no viewers too. Before making the project public, let's proceed with caution and add "you" and "anotherperson" as a viewer, for these specific version. We'll use the `--verbose` option to ask what permissions will be applied as a result, showing what we had before and what we'll get after:

```
adb> permissions set PRJ000000021@1 --add-viewers=you,anotherperson --verbose
Authenticating user: 'lelongs'.
Successfully authenticated.
Merging with existing permissions
Existing permissions found:
owners:
- lelongs
read_access: viewers
scope: project
viewers: null
write_access: owners

New permissions to apply:
```

```
owners:
- lelongs
read_access: viewers
scope: version
viewers:
- anotherperson
- you
write_access: owners

Replace existing permissions? [y/n] (n): y
Authenticating user: 'lelongs'.
Successfully authenticated.
Overwriting existing context 'olympus-api-2-uat'
Indexing job created, new permissions will be active once done:
job_id: b3dd109f-35b6-4223-a6e9-4dc81f3aed23
job_url: http://democli.api.artifactdb.io/v1/jobs/b3dd109f-35b6-4223-a6e9-4dc81f3aed23
path: /jobs/b3dd109f-35b6-4223-a6e9-4dc81f3aed23
status: accepted
```

We can see that before the scope was `project`. Because we specified a version with `PRJ000000021@1`, we declare version-level permissions. We now also have a list of viewers, and finally, we haven't specified the owners and the rest of the fields, but the CLI used the existing permissions (at project-level) to complete the rest of the profile.

Fetching permissions should now display the following. Notice the project-level permissions have not been modified.

```
adb> permissions show PRJ000000021@1
Authenticating user: 'lelongs'.
Successfully authenticated.
owners:
- lelongs
read_access: viewers
scope: version
viewers:
- anotherperson
- you
write_access: owners
```

```
adb> permissions show PRJ000000021
Authenticating user: 'lelongs'.
Successfully authenticated.
owners:
- lelongs
read_access: viewers
scope: project
viewers: null
write_access: owners
```

Let's switch the read access to public for that project. We can either specify `--read-access: public` or the shortcut `--public`, this is equivalent:

```
adb> permissions set PRJ000000021 --public --verbose
Authenticating user: 'lelongs'.
Successfully authenticated.
Merging with existing permissions
Existing permissions found:
owners:
```

```
- lelongs
read_access: viewers
scope: project
viewers: null
write_access: owners

New permissions to apply:
owners:
- lelongs
read_access: public
scope: project
write_access: owners

Replace existing permissions? [y/n] (n): y
```

Once permissions are applied (the instance reindexed the project), we can access the artifacts anonymously, since it's public. We can turn off the authentication and switch to anonymous access, using the `logout` command.

```
adb> logout
Anonymous access enabled
```

```
adb> search PRJ000000021 --load=my_search_profile
No results
adb>
```

What? There's no results, why? We set the permissions specifically for version 1, and then enable public access at the project level. Version-level permissions have precedence, so our artifacts are still private. We could set public access for the version, but let's remove these specific permissions scoped for it, it's been confusing. We'll need to `login` again, since we need to be an owner to change permissions.

```
adb> permissions delete PRJ000000021@1
Unable to fetch permissions: 401 - Not authenticated: '{'status': 'error', 'reason': 'Not
    authenticated'}'
No existing permissions found, nothing to do
```

```
adb> login
Authenticating user: 'lelongs'.
Successfully authenticated.
Authenticated access enabled for context 'olympus-api-2-uat'
adb>
```

```
adb> permissions delete PRJ000000021@1
Authenticating user: 'lelongs'.
Successfully authenticated.
Are you sure you want to delete these permissions? [y/n] (n): y
Authenticating user: 'lelongs'.
Successfully authenticated.
Overwriting existing context 'olympus-api-2-uat'
Indexing job created, new inherited permissions will be active once done:
job_id: 16b46da9-3b32-4159-9e97-adda81b1665d
job_url: http://democli.api.artifactdb.io/v1/jobs/16b46da9-3b32-4159-9e97-adda81b1665d
path: /jobs/16b46da9-3b32-4159-9e97-adda81b1665d
status: accepted
```

The permissions profile was removed, and the version reindexed. It now inherits from the permissions

defined for the whole project, that is, public.

```
adb> logout
Anonymous access enabled
```

```
adb> search PRJ000000021 --load=my_search_profile
_extra:
  id: PRJ000000021:file1.txt@1
path: file1.txt

---
_extra:
  id: PRJ000000021:file2.txt@1
path: file2.txt

---
No more results
```

All good now!

Epilogue

We did a great job, but let's face it, looking at the artifacts again, it's pretty clear there's little value. While we can't delete a project by default in ArtifactDB (mostly to make sure there's no accidental data loss, but also for audit purpose), we can hide it.

```
adb> login
Authenticating user: 'lelongs'.
Successfully authenticated.
Authenticated access enabled for context 'olympus-api-2-uat'
adb>
adb> permissions set PRJ000000021 --hide
Authenticating user: 'lelongs'.
Successfully authenticated.
Merging with existing permissions
After applying permissions, the project  (or version) will be hidden and unaccessible for
    users (except admins)
Are you sure you want to hide this project/version? [y/n] (n): y
Replace existing permissions? [y/n] (n): y
Authenticating user: 'lelongs'.
Successfully authenticated.
Overwriting existing context 'olympus-api-2-uat'
Indexing job created, new permissions will be active once done:
job_id: 7c281197-0bae-4e0c-bc8b-71a4533ab4d6
job_url: http://democli.api.artifactdb.io/v1/jobs/7c281197-0bae-4e0c-bc8b-71a4533ab4d6
path: /jobs/7c281197-0bae-4e0c-bc8b-71a4533ab4d6
status: accepted
...
```

The CLI notices the intent, and ask for confirmation. Once the job is done, we can try to search for the project again, as an authenticated user and former owner:

```
adb> search PRJ000000021 --load=my_search_profile
No results
```

Gone… Indeed, after that operation, no one will be able to access the project anymore, not even us[5]. Only an admin would be able to change the permissions and restore visiblity to the project.

Did we shoot ourselve in the foot? Yes, but that's the end of this tutorial, so I guess it's fine…

[5] We could do `permissions set PRJ000000021 --read-acces=none` if keep visibility for us, but here we really want to get rid of that project.

Design

The ArtifactDB CLI heavily relies on a python generic implementation of an ArtifactDB client, conveniently named artifactdb-client. This package handles all the low level access to the REST API, the authentication, uploading logic, etc…

The client itself is based on a design allowing different components to be registered and add more features. The uploader component is one example, providing different ways to upload data to ArtifactDB: using S3 presigned URLs, or STS credentials with a specific implementation to interface with S3 (boto3, awscli, etc…). As a consequence, the features available on the CLI side directly depends on what's available on the client side.

The implementation of the CLI is based on typer, by author Tiangolo, the same person who wrote FastAPI. This library enables quick CLI implementation, with clear documentation. It is also using rich behind the scene, for terminal output formatting, and so does the CLI itself.

The CLI provides core commands, such `context`, `upload`, etc… but it can be extended using a plugin architecture. The terminal plugin is one example[1]. Once registered using the command `plugin add`, the CLI discovers automatically the additional commands declared in the plugins. Refer to the command reference for more about managing CLI plugins. A developer tutorial will be available later, to show how to implement a plugin and extend the CLI functionalities.

[1] The "terminal" plugin adds command to manage a terminal, based on wetty, to access an admin pod from a web brower, to perform administrative tasks not available from the REST API.

FAQ

How can I disable the colors in output? I prefer monochrome text...

It is sad. But colors can be disabled with by setting the environment variable `NO_COLOR`:

```
$ export NO_COLOR=1
$ adb context show
...
# sad and boring monochrome output
```

Is ArtifactDB CLI thread-safe?

No, it is not. It should not be used in environments where concurrent usage could happen, mostly because the CLI writes and updates configuration files while it's being used, and these write accesses are not protected from concurrent writing in current implementation.

Is it true that the ArtifactDB CLI is the best thing that happened to humanity in a long while?

Yes.

Command reference

Usage:

```
$ adb [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--install-completion`: Install completion for the current shell.
- `--show-completion`: Show completion for the current shell, to copy it or customize the installation.
- `--help`: Show this message and exit.

Commands:

- `context`: Manage ArtifactDB contexts (connections,…
- `download`: Download artifacts.
- `job`: Manage jobs (indexing, …)
- `login`: Switch to authenticated access.
- `logout`: Switch to anonymous access.
- `permissions`: Manage project's permissions
- `plugins`: Manage CLI plugins
- `search`: Searching metadata documents, using active…
- `shell`: Launch interactive shell
- `tasks`: Manage backend tasks (core & plugins).
- `upload`: Upload artifacts.
- `version`: Print the CLI version information

`adb context`

Manage ArtifactDB contexts (connections, clients, …)

Usage:

```
$ adb context [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `create`: Create a new ArtifactDB context with…
- `list`: List available ArtifactDB contexts
- `show`: Show ArtifactDB context details
- `use`: Set given context as current one

`adb context create`

Create a new ArtifactDB context with connection details.

Usage:

```
$ adb context create [OPTIONS] URL
```

Arguments:

- `URL`: URL pointing to the REST API root endpoint [required]

Options:

- `--auth-url TEXT`: Keycloak auth URL (contains realm name, eg. `awesome` https://mykeycloak.mycompany.
- `--name TEXT`: Context name. The instance name (if exposed) is used by default to name the context
- `--auth-client-id TEXT`: Client ID used for authentication. The instance's main client ID (if exposed) is used by default
- `--auth-username TEXT`: Username used in authentication, default to `whoami`
- `--project-prefix TEXT`: Project prefix used in that context. If the instance exposes that information, a selection can be made, otherwise, the instance's default is used
- `--auth-service-account-id TEXT`: Create a context for a service account, instead of current user
- `--force / --no-force`: Don't ask for confirmation before creating the context [default: no-force]
- `--help`: Show this message and exit.

`adb context list`

List available ArtifactDB contexts

Usage:

```
$ adb context list [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

`adb context show`

Show ArtifactDB context details

Usage:

```
$ adb context show [OPTIONS] [NAME]
```

Arguments:

- `[NAME]`: Context name (or current one if not specified)

Options:

- `--help`: Show this message and exit.

`adb context use`

Set given context as current one

Usage:

```
$ adb context use [OPTIONS] NAME
```

Arguments:

- `NAME`: Context name [required]

Options:

- `--help`: Show this message and exit.

`adb download`

Download artifacts.

Usage:

```
$ adb download [OPTIONS] [WHAT] [DEST]
```

Arguments:

- `[WHAT]`: Download given artifact(s). Use [project_id] to download all files of the latest version for a given project, [project_id@version] for a specific version, or an ArtifactDB ID [project:path@version] for a single artifact. Alternately, –project-id, –version and –id options can be used to achieve the same result.
- `[DEST]`: Path to folder containing the files to download, defaulting to current folder. [default: .]

Options:

- `--project-id TEXT`: Download data from given project ID.
- `--version TEXT`: Requires –project-id. Download specific version of a project, or the latest available if omitted
- `--id TEXT`: ArtifactDB ID representing the file to download. Must not be used with –project-id and –version options
- `--cache TEXT`: Cache mode used to cache files while downloaded. Default is no cache
- `--verbose / --no-verbose`: Print information about what the command is performing [default: no-verbose]
- `--overwrite / --no-overwrite`: If local files exist, don't overwrite with downloaded artifact. [default: no-overwrite]
- `--help`: Show this message and exit.

adb job

Manage jobs (indexing, …)

Usage:

```
$ adb job [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `check`: Using active context, check status for all…
- `list`: List all jobs recorded in current context,…

`adb job check`

Using active context, check status for all jobs, or given job ID. Jobs statuses are updated each they're checked.

Usage:

```
$ adb job check [OPTIONS] [JOB_ID]
```

Arguments:

- `[JOB_ID]`: Job ID (all jobs checked if omitted). Job ID can be one recorded in the context, or a manual entry.

Options:

- `--format [json|yaml|human]`: Return job status in specified format, default is human-readable [default: human]
- `--prune [terminated|success|failure|pending|none|all|purged]`: Prune jobs with given status after reporting it [default: success]
- `--verbose / --no-verbose`: Display additional information about jobs (eg. traceback, etc…) [default: no-verbose]
- `--help`: Show this message and exit.

`adb job list`

List all jobs recorded in current context, with last checked status.

Usage:

```
$ adb job list [OPTIONS]
```

Options:

- `--verbose / --no-verbose`: Print all jobs information [default: no-verbose]
- `--help`: Show this message and exit.

`adb login`

Switch to authenticated access. Only effective within an ArtifactDB shell.

Usage:

```
$ adb login [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

adb logout

Switch to anonymous access. Only effective within ArtifactDB shell.

Usage:

```
$ adb logout [OPTIONS]
```

Options:

- `--purge / --no-purge`: Logout and delete cached credentials. Subsequent login will trigger authentication again. [default: no-purge]
- `--help`: Show this message and exit.

adb permissions

Manage project's permissions

Usage:

```
$ adb permissions [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `delete`: Delete permission profile for a given…
- `set`: Replace existing permissions or create new…
- `show`: Show current permissions for a given…

adb permissions delete

Delete permission profile for a given project/version. After deletion, the new permissions will inherit from upper scope: version > project > global. If no permissions can be inherited, the project/version becomes permanently unavailable (except admins), USE WITH CAUTION ! (you've been warned)

Usage:

```
$ adb permissions delete [OPTIONS] [WHAT]
```

Arguments:

- `[WHAT]`: Identifier for which the permissions will deleted. Notation can be a [project_id], [project_id@version] for a specific version. Alternately, –project-id and –version can be used.

Options:

- `--project-id TEXT`: Project ID.
- `--version TEXT`: Requires –project-id. Delete permissions for a specific version of a project
- `--confirm / --no-confirm`: Ask for confirmation before repla. [default: confirm]
- `--verbose / --no-verbose`: Show permissions that will be deleted [default: no-verbose]
- `--help`: Show this message and exit.

`adb permissions set`

Replace existing permissions or create new ones. A full permissions document can be passed with `--permissions`, or individual parts can be provided with the other options.

Usage:

```
$ adb permissions set [OPTIONS] [WHAT]
```

Arguments:

- `[WHAT]`: Identifier for which the permissions will be replaced or set. Notation can be a [project_id], [project_id@version] for a specific version. Alternately, –project-id and –version can be used.

Options:

- `--project-id TEXT`: Project ID.
- `--version TEXT`: Requires –project-id. Fetch permissions for a specific version of a project
- `--permissions TEXT`: New permissions, JSON string format. Partial permissions information will be completed with default permissions values. See also –merge.
- `--merge / --no-merge`: Using existing permissions as base, and merge new declared permissions on top of it. This allows to change parts of the permissions profile without having to re-declare it completely [default: merge]
- `--read-access TEXT`: Defines read access rule
- `--write-access TEXT`: Defines write access rule
- `--viewers TEXT`: Replace existing viewers with comma-separated list of new viewers. An empty string remove all viewers.

- `--add-viewers TEXT`: Add one or more viewers (comma-separated) to existing ones
- `--owners TEXT`: Replace existing owners with comma-separated list of new owners. An empty string remove all owners.
- `--add-owners TEXT`: Add one or more owners (comma-separated) to existing ones
- `--public / --no-public`: Make the project publicly accessible (shortcut to –read-access=public [default: no-public]
- `--private / --no-private`: Restrict the access to the project to viewers only (shortcut to –read-access=viewers [default: no-private]
- `--hide / --no-hide`: Hide the dataset to anyone except admins (shortcut to –read-access=none –write-access=none [default: no-hide]
- `--confirm / --no-confirm`: Ask for confirmation if existing permissions exist, before replacing them. [default: confirm]
- `--verbose / --no-verbose`: Show additional information, eg. existing vs. new permissions, etc… [default: no-verbose]
- `--help`: Show this message and exit.

`adb permissions show`

Show current permissions for a given project, version. Note permissions for a specific version may inherit from the project itself, check the `scope` field to determine is permissions are project or version specific.

Usage:

```
$ adb permissions show [OPTIONS] [WHAT]
```

Arguments:

- `[WHAT]`: Identifier used to obtain current permissions from. Notation can be a [project_id], [project_id@version] for a specific version. Alternately, –project-id and –version can be used.

Options:

- `--project-id TEXT`: Project ID.
- `--version TEXT`: Requires –project-id. Fetch permissions for a specific version of a project
- `--help`: Show this message and exit.

`adb plugins`

Manage CLI plugins

Usage:

```
$ adb plugins [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `add`
- `disable`: Disable a registered plugin.
- `enable`: Enable a registered plugin.
- `list`: List registered plugins.
- `remove`
- `show`: Show plugin configuration.

`adb plugins add`

Usage:

```
$ adb plugins add [OPTIONS] NAME
```

Arguments:

- `NAME`: Plugin name to install [required]

Options:

- `--location TEXT`: PyPI index URL (default: pip's default one, https://pypi.org/simple), or local folder
- `--verbose / --no-verbose`: Print debug information while registering the plugin. [default: no-verbose]
- `--help`: Show this message and exit.

`adb plugins disable`

Disable a registered plugin. Useful to deactivate a plugin causing issues.

Usage:

```
$ adb plugins disable [OPTIONS] NAME
```

Arguments:

- NAME: Plugin name [required]

Options:

- --help: Show this message and exit.

adb plugins enable

Enable a registered plugin.

Usage:

```
$ adb plugins enable [OPTIONS] NAME
```

Arguments:

- NAME: Plugin name [required]

Options:

- --help: Show this message and exit.

adb plugins list

List registered plugins.

Usage:

```
$ adb plugins list [OPTIONS]
```

Options:

- --help: Show this message and exit.

adb plugins remove

Usage:

```
$ adb plugins remove [OPTIONS] NAME
```

Arguments:

- NAME: Plugin name to install [required]

Options:

- `--verbose / --no-verbose`: Print debug information while registering the plugin. [default: no-verbose]
- `--help`: Show this message and exit.

`adb plugins show`

Show plugin configuration.

Usage:

```
$ adb plugins show [OPTIONS] NAME
```

Arguments:

- `NAME`: Plugin name [required]

Options:

- `--help`: Show this message and exit.

`adb search`

Searching metadata documents, using active context.

Usage:

```
$ adb search [OPTIONS] [QUERY]
```

Arguments:

- `[QUERY]`: ElasticSearch query string. Ex: `path:myfile.txt AND title:"important"`

Options:

- `--fields TEXT`: Comma separated list of fields to display in the search results. Dot-field notation can be use to refer to an inner field, eg. `_extra.permissions.owners`
- `--project-id TEXT`: Search within a specific project ID. Same as specifying `_extra.project_id:<project_id>` in the query parameter.
- `--version TEXT`: Requires –project-id. Searching within a specific version. Same as specifying `_extra.version:<version>` in the query parameter.
- `--latest / --no-latest`: Search for latest versions only [default: no-latest]
- `--size INTEGER RANGE`: Number of results returned in a page [1<=x<=100]

- `--format TEXT`: Format used to display results. Default is YAML format.
- `--save TEXT`: Save search parameters in a profile
- `--load TEXT`: Load a saved search profile and use it for search parameters.
- `--delete TEXT`: Delete a search profile
- `--ls / --no-ls`: List search profile names and exit. [default: no-ls]
- `--show TEXT`: Show search profile content and exit.
- `--verbose / --no-verbose`: Print more informational/debug messages [default: no-verbose]
- `--help`: Show this message and exit.

adb shell

Launch interactive shell

Usage:

```
$ adb shell [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

adb tasks

Manage backend tasks (core & plugins). Note: most commands required admin permissions.

Usage:

```
$ adb tasks [OPTIONS] COMMAND [ARGS]...
```

Options:

- `--help`: Show this message and exit.

Commands:

- `list`: List registered backend tasks
- `logs`: Show logs for all recent tasks execution.
- `run`: Trigger the execution of a given task,...
- `show`: Show task information (arguments, etc...)

`adb tasks list`

List registered backend tasks

Usage:

```
$ adb tasks list [OPTIONS]
```

Options:

- `--type [core|plugin]`: List tasks with given type
- `--help`: Show this message and exit.

`adb tasks logs`

Show logs for all recent tasks execution.

Usage:

```
$ adb tasks logs [OPTIONS] [NAME]
```

Arguments:

- `[NAME]`: Show logs for a given task

Options:

- `--clear / --no-clear`: Clear cache storing recent task execution logs. [default: no-clear]
- `--help`: Show this message and exit.

`adb tasks run`

Trigger the execution of a given task, with its parameters (if any).

Usage:

```
$ adb tasks run [OPTIONS] NAME
```

Arguments:

- `NAME`: Name of the task to trigger. [required]

Options:

- `--params TEXT`: JSON string representing the named params to pass for the execution. Ex: ‘{“param1”: “value1”, “param2”: false, “param3”: [1,2,3]}’
- `--help`: Show this message and exit.

`adb tasks show`

Show task information (arguments, etc…)

Usage:

```
$ adb tasks show [OPTIONS] [NAME]
```

Arguments:

- `[NAME]`: Name of the task

Options:

- `--help`: Show this message and exit.

`adb upload`

Upload artifacts.

Usage:

```
$ adb upload [OPTIONS] STAGING_DIR
```

Arguments:

- `STAGING_DIR`: Path to folder containing the files to upload [required]

Options:

- `--project-id TEXT`: Upload data as a new version within an existing project. Creating a new version requires ownership permissions on the existing project. When not set (default) a new project is created.
- `--version TEXT`: Requires –project-id. Upload data as a new version, specified with this option. Setting a specific version usually requires extra permisions, as this use case is rare and dangerous…
- `--owners TEXT`: Owner(s) of the uploaded artifacts. Use comma , to specify more than one.Defaults to authenticated user or service account.
- `--viewers TEXT`: Viewers(s) of the uploaded artifacts. Use comma , to specify more than one.
- `--read-access [owners|viewers|authenticated|public|none]`: Role to allow data access in read-only mode. `viewers` restricts access to the list specified with the argument –viewers (default), same for `owners`. `authenticated` allows read-only access

to any users with a valid token, `public` allows anonymous access. `none` disables read-only access. [default: viewers]

- `--write-access [owners|viewers|authenticated|public|none]`: Role to allow data access in read-write mode. `owners` restricts read/write access to the list specified with the argument –owners, same for `viewers`. `authenticated` allows read/write access to any users with a valid token, `public` allows any anonymous users to modify data, and `none` disable read/write access. [default: owners]
- `--permissions-json TEXT`: Permissions for the newly creation project or version, in JSON format. See documentation for the context
- `--upload-mode [presigned|sts:boto3]`: Method used to upload data. `presigned` uses S3 presigned-URLs for each file to upload (recommended only for small files, max 5GiB/file). `sts:*` uses STS credentials, with a specific client implementation, eg `boto3`, `awscli`, `s5cmd`, etc… depending on what is available. STS credentialsenables multipart/parallel upload, and file size up to 5TB/file. [default: presigned]
- `--expires-in TEXT`: Upload transient artifacts, expiring (purged) after given experation date. Ex: '2022-12-25T00:00:00', 'December 25th', 'in 3 days', etc…
- `--validate / --no-validate`: Validate metadata JSON files, using the $schema field and API validation endpoint [default: validate]
- `--verbose / --no-verbose`: Print information about what the command is performing [default: no-verbose]
- `--confirm / --no-confirm`: Ask for confirmation before proceeding with the upload [default: no-confirm]
- `--help`: Show this message and exit.

`adb version`

Print the CLI version information

Usage:

```
$ adb version [OPTIONS]
```

Options:

- `--help`: Show this message and exit.

Appendix

Revision history

- 2023-02-03:
  - initial release
- 2023-02-16:
  - custom CA certificates installation instructions
  - improve formatting in code block
  - minor update on command reference