dtool Documentation

Release 3.15.0

Tjelvar Olsson

Apr 26, 2019

Contents

Make your data more resilient, portable and easy to work with by packaging files & metadata into self contained datasets.

CHAPTER 1

dtool: Manage Scientific Data

Make your data more resilient, portable and easy to work with by packaging files & metadata into self contained datasets.

- Documentation: http://dtool.readthedocs.io
- Paper: https://doi.org/10.7717/peerj.6562
- Free software: MIT License

1.1 Overview

dtool is a suite of software for managing scientific data and making it accessible programatically. It consists of a command line interface `dtool` and a Python API: dtoolcore.

The `dtool` command line interface allows one to organise files into datasets and to move datasets between different storage solutions, for example from local disk to remote object storage. Importantly it also provides methods to verify that the transfer has been successful.

The Python API gives complete access to the data and metadata in a dataset. It makes it easy to create scripts for processing the items, or a subset of items, in a dataset. The Python API also allows datasets to be constructed programatically.

dtool is extensible, meaning that it is possible to create plugins both for adding functionality to the command line interface and for creating interfaces to custom storage backends.

The `dtool` Python package is a meta package that installs the packages:

- dtoolcore - core API
- dtool-cli - CLI plugin scaffold
- dtool-config - CLI commands for configuring dtool

- dtool-create - CLI commands for creating datasets
- dtool-info - CLI commands for getting information about datasets
- dtool-symlink - storage broker interface allowing symlinking to data
- dtool-http - storage broker interface allowing read only access to datasets over HTTP

Installation:

```
$ pip install dtool
```

There are support packages for several object storage solutions:

- dtool-s3 - storage broker interface to S3 object storage
- dtool-azure - storage broker interface to Azure Storage
- dtool-ecs - storage broker interface to ECS S3 object storage
- dtool-irods - storage broker interface to iRODS

If you have access to Amazon S3, Microsoft Azure, ECS S3 or iRODS storage you may also want to install support for these:

```
$ pip install dtool-s3 dtool-azure dtool-ecs dtool-irods
```

Usage:

```
$ dtool create my-awesome-dataset
Created proto dataset file:///Users/olssont/my-awesome-dataset
Next steps:
1. Add raw data, eg:
   dtool add item my_file.txt file:///Users/olssont/my-awesome-dataset
   Or use your system commands, e.g:
   mv my_data_directory /Users/olssont/my-awesome-dataset/data/
2. Add descriptive metadata, e.g:
   dtool readme interactive file:///Users/olssont/my-awesome-dataset
3. Convert the proto dataset into a dataset:
   dtool freeze file:///Users/olssont/my-awesome-dataset
```

CHAPTER 2

Installation notes

dtool is a Python package that is pip installable.

Make sure that `pip`, `setputools` and `wheel` are up to date. This is a requirement of one of the dependencies (`ruamel.yaml`).

```
$ pip install -U pip setuptools wheel
```

dtool can then be installed using `pip`.

```
$ pip install dtool
```

2.1 Adding support for S3 object storage

Install the `dtool-s3` package using `pip`.

```
$ pip install dtool-s3
```

To configure Amazon S3 credentials see the README file in the dtool-s3 GitHub repository.

2.2 Adding support for Azure storage

Install the `dtool-azure` package using `pip`.

```
$ pip install dtool-azure
```

To configure Microsoft Azure credentials see the README file in the dtool-azure GitHub repository.

2.3 Adding support for ECS S3 object storage

Install the `dtool-ecs` package using `pip`.

```
$ pip install dtool-ecs
```

To configure ECS S3 object storage credentials see the README file in the dtool-ecs GitHub repository.

2.4 Adding support for iRODS storage

Install the `dtool-irods` package using `pip`.

```
$ pip install dtool-irods
```

Warning: In order to be able to use the iRODS backend storage you will need to install the iCommands. Linux packages can be downloaded from irods.org/download. On Mac OSX these can be installed using the brew package manager:

```
$ brew install irods
```

For more details see the dtool-irods GitHub repository.

CHAPTER 3

Philosophy - what is dtool?

3.1 What problem is dtool solving?

Managing data as a collection of individual files is hard. Analysing that data will require that certain sets of files are present, understanding it requires suitable metadata, and copying or moving it while keeping its integrity is difficult.

dtool solves this problem by packaging a collection of files and accompanying metadata into a self contained and unified whole: a dataset.

Having metadata separate from the data, for example in an Excel spread sheet with links to the data files, it becomes difficult to reorganise the data without fear of breaking links between the data and the metadata. By encapsulating both the data files and associated metadata in a dataset one is free to move the dataset around at will. The high level organisation of datasets can therefore evolve over time as data management processes change.

dtool also solves an issue of trust. By including file hashes as metadata it is possible to verify the integrity of a dataset after it has been moved to a new location or when coming back to a dataset after a period of time.

It is possible to discover and access both metadata and data files in a dataset. It is therefore easy to create scripts and pipelines to process the items, or a subset of items, in a dataset.

3.2 What is a "dtool dataset"?

Briefly, a dtool dataset consists of:

- The files added to the dataset, known as the dataset "items"
- Metadata used to describe the dataset as a whole
- Metadata describing the items in the dataset

The exact details of how this data and metadata is stored depends on the "backend" (the type of storage used). In other words a dataset is stored differently on local file system disk to how it is stored in Amazon S3 object store. However, the `dtool` commands and the Python API for interacting with datasets are the same for all backends.

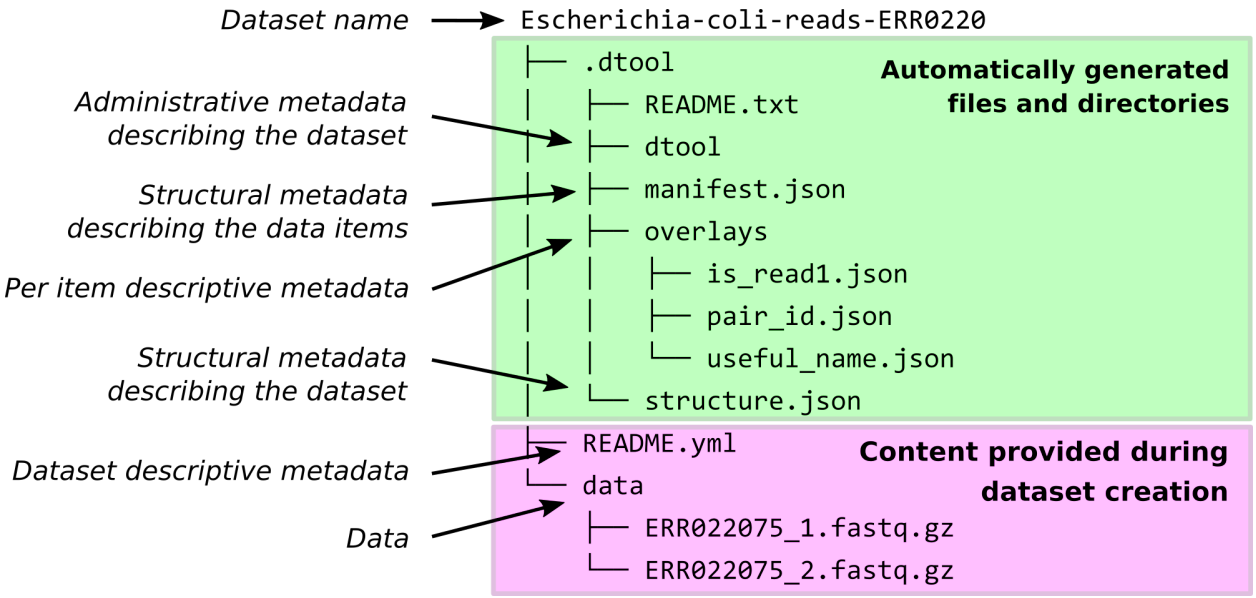
3.3 What does a dtool dataset look like on local disk?

Below is the structure of a fictional dataset containing three items from an RNA sequencing experiment.

```
$ tree ~/my_dataset
/Users/olssont/my_dataset
├── README.yml
└── data
    ├── rna_seq_reads_1.fq.gz
    ├── rna_seq_reads_2.fq.gz
    └── rna_seq_reads_3.fq.gz
```

The `README.yml` file is where metadata used to describe the whole dataset is stored. The items of the dataset are stored in the directory named `data`.

There is also hidden metadata, stored as plain text files, in a directory named `.dtool`. This should not be edited directly by the user.



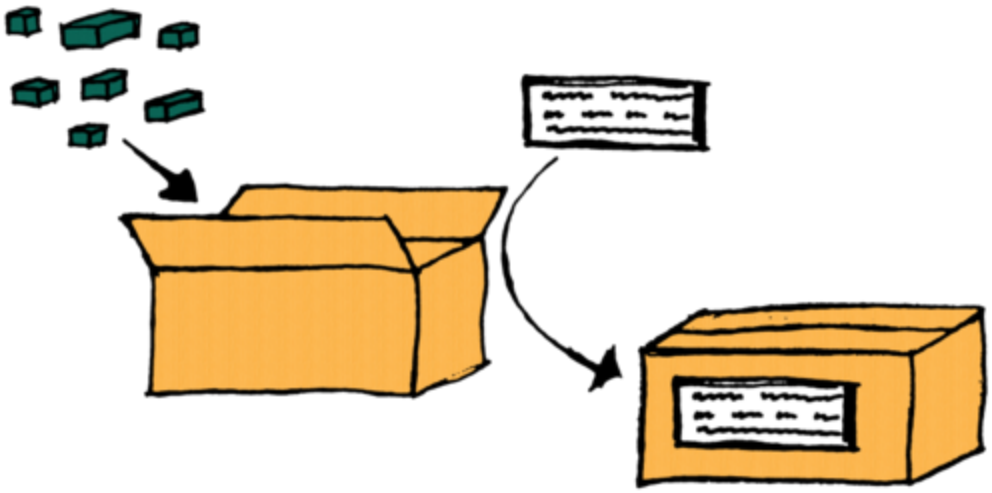
3.4 How does one create a dtool dataset?

This happens in stages:

1. One creates a so called "proto dataset"
2. One adds data and metadata to this proto dataset
3. One converts the proto dataset into a dataset by "freezing" it

Once a proto dataset is "frozen" it is simply referred to as a dataset and it is no longer possible to modify the data in it. In other words it is not possible to add or remove items from a dataset or to alter any of the items in a dataset.

The process can be likened to creating an open box (the proto dataset), putting items (data) into it, sticking a label (metadata) on it, and closing the box (freezing the dataset).



3.5 Give me more details!

An in depth discussion of dtool can be found in the paper Lightweight data management with dtool.

CHAPTER 4

Quick start guide

This quick start guide shows how the `dtool` command line tool can be used to accomplish some common data management tasks.

4.1 Organising files into a dataset on local disk

In this scenario one simply wants to organise one or more files into a dataset in the file system on the local computer.

When working on local disk a dataset is simply a standardised directory layout combined with some hidden files used to annotate the dataset and its items.

The first step is to create a "proto" dataset. The command below creates a dataset named `fishers-iris-data` in the current working directory.

```
$ dtool create fishers-iris-data
```

One can now add files to the dataset by moving/copying them to the `fisher-iris-data/data` directory, or by using the built in `dtool add item` command. In the example below the file `iris.csv` is added to the proto dataset.

```
$ touch iris.csv
$ dtool add item iris.csv fishers-iris-data
```

Metadata describing the data is as important as the data itself. Metadata describing the dataset is stored in the file `fisers-iris-data/README.yml`. An easy way to add content to this file is to use the `dtool readme interactive`, which will prompt for input regarding the dataset.

```
$ dtool readme interactive fishers-iris-data
description [Dataset description]: Fisher's classic iris data, but with an empty file␣
↪:(
project [Project name]: dtool demo
confidential [False]:
personally_identifiable_information [False]:
```

(continues on next page)

(continued from previous page)

```
name [Your Name]: Tjelvar Olsson
email [olssont@nbi.ac.uk]:
username [olssont]:
creation_date [2017-10-06]:
Updated readme
To edit the readme using your default editor:
dtool readme edit fiser-iris-data
```

Finally, to convert the proto dataset into a dataset one uses the `dtool freeze` command.

```
$ dtool freeze fishers-iris-data
Generating manifest  [####################################]  100%  iris.csv
Dataset frozen fiser-iris-data
```

4.2 Copying data from an external hard drive to remote storage as a dataset

Genome sequencing generates large volumes of data, which are often sent from the sequencing company to the user by posting an external hard drive. When backing up such data on a remote storage system one does not want to have to reorganise the data before copying it to the remote storage system.

In this case one can create a "symlink" dataset and copy that to the remote storage. A symlink dataset is a dataset where the data directory is a symlink to another location, for example the data directory on the external hard drive.

```
$ dtool create bgi-sequencing-12345 --symlink-path /mnt/external-hard-drive
```

Again, adding metadata to the dataset is vital.

```
$ dtool readme interactive bgi-sequencing-12345
```

One can then convert the proto dataset into a dataset by "freezing" it.

```
$ dtool freeze bgi-sequencing-12345
```

It is now time to copy the dataset to the remote storage. The command below assumes that one has credentials setup to write to the Amazon S3 bucket `dtool-demo`. The command copies the local dataset to the S3 `dtool-demo` bucket.

```
$ dtool cp bgi-sequencing-12345 s3://dtool-demo/
```

The command above returns feedback on the URI used to identify the dataset in the remote storage. In this case `s3://dtool-demo/1e47c076-2eb0-43b2-b219-fc7d419f1f16`.

The URI used to identify the dataset uses the UUID of the dataset rather than the dataset's name. This is to avoid name clashes in the object storage.

Finally, one may want to confirm that the data transfer was successful. This can be achieved using the `dtool diff` command, which should show no differences if the transfer was successful.

```
$ dtool diff bgi-sequencing-12345 s3://dtool-demo/1e47c076-2eb0-43b2-b219-fc7d419f1f16
```

By default only identifiers and file sizes are compared. To check file hashes make use of the `--full` option.

Warning: When comparing datasets identifiers, sizes and hashes are compared. When checking that the hashes are identical the hashes for the first dataset are recalculated using the hashing algorithm of the reference dataset (the second). If the dataset in S3 had been specified as the first argument then all the files would have had to have been downloaded to the local disk before calculating their hashes, which would have made the command slower.

4.3 Copying a dataset from remote storage to local disk

After having copied a dataset to a remote storage system one may have deleted the copy on the local disk. In this case one may want to be able to get the dataset back onto the local disk.

This can be achieved using the `dtool cp` command. The command below copies the dataset in iRODS to the current working directory.

```
$ dtool cp s3://dtool-demo/1e47c076-2eb0-43b2-b219-fc7d419f1f16 ./
```

Note that on the local disk the dataset will use the name of the dataset rather than the UUID, in this example `bgi-sequencing-12345`.

Again one can verify the data transfer using the `dtool diff` command.

```
$ dtool diff bgi-sequencing-12345 s3://dtool-demo/1e47c076-2eb0-43b2-b219-fc7d419f1f16
```

CHAPTER 5

Working with datasets

5.1 Listing datasets

It is possible to list all datasets in a directory or in a S3 bucket using the `dtool ls` command.

```
$ dtool ls ~/my_datasets
bgi-sequencing-12345
  file:///Users/olssont/my_datasets/bgi-sequencing-12345
drone-images
  file:///Users/olssont/my_datasets/drone-images
fishers-iris-data
  file:///Users/olssont/my_datasets/fishers-iris-data
my_rnaseq_data
  file:///Users/olssont/my_datasets/my_rnaseq_data
```

Tip: When using this command proto datasets are highlighted in red.

Tip: The `dtool ls` command takes a URI. As such it can be used to list the datasets in remote storage locations. The example below lists all the datasets in the S3 bucket named `dtool-demo`:

```
$ dtool ls s3://dtool-demo/
```

5.2 Generating an inventory of datasets

It is possible to generate CSV/TSV/HTML inventories of datasets in a directory or in another base URI such as an Amazon S3 bucket. For example, the command below is used to generate a HTML report of all the datasets in the s3://dtool-demo/ bucket.

```
$ dtool inventory --format html s3://dtool-demo/ > inventory.html
```

5.3 Verifying a dataset has not been modified since freezing it

A dtool dataset has metadata listing its items and their hashes. This information can be used to verify that a dataset is in the same state as it was when it was frozen.

In the example below the dataset has been corrupted in three ways.

1. The file `rna_seq_reads_4.fq.gz` has been added to it
2. The file `rna_seq_reads_3.fq.gz` has been deleted from it
3. The content of the file `rna_seq_reads_1.fq.gz` has been modified

```
$ dtool verify ~/my_datasets/my_rnaseq_data
Unknown item: 49919bdae83011b96bf54d984735e24c4419feb5 rna_seq_reads_4.fq.gz
Missing item: 72b24007759c0086a316d13838021c2571853a16 rna_seq_reads_3.fq.gz
```

By default only identifiers and file sizes are compared. To check file hashes make use of the `--full` option.

```
$ dtool verify --full ~/my_datasets/my_rnaseq_data
Unknown item: 49919bdae83011b96bf54d984735e24c4419feb5 rna_seq_reads_4.fq.gz
Missing item: 72b24007759c0086a316d13838021c2571853a16 rna_seq_reads_3.fq.gz
Altered item: d4e065787eab480e9cbd2bac6988bc7717464c83 rna_seq_reads_1.fq.gz
```

5.4 Displaying the README descriptive metadata

To display the README metadata used to describe the dataset one can make use of the `dtool readme show` command.

```
$ dtool readme show ~/my_datasets/chrX-rna-seq
---
description: RNA-seq sample data
creation_date: 2017-11-20
ftp: "ftp://ftp.ccb.jhu.edu/pub/RNAseq_protocol/"
doi: "10.1038/nprot.2016.095"
```

5.5 Reporting summary information about a dataset

One often wants to find out how many items are in a dataset and what their total size is. This can be achieved using the `dtool summary` command.

```
$ dtool summary ~/my_datasets/drone-images
name: drone-images
uuid: c2542c2b-d149-4f73-84bc-741bf9af918f
creator_username: hartleym
number_of_items: 59
size: 152.5MiB
frozen_at: 2017-09-19
```

5.6 Listing the item identifiers in a dataset

To list all the item identifiers in a dataset one can use the `dtool identifiers` command.

```
$ dtool identifiers ~/my_datasets/my_rnaseq_data
b0f92a668d24a3015692b0869e2b7590a62a380c
72b24007759c0086a316d13838021c2571853a16
d4e065787eab480e9cbd2bac6988bc7717464c83
```

Tip: Using `dtool ls` on a dataset URI results in a list of item identifiers and relapths:

```
$ dtool ls ~/my_datasets/my_rnaseq_data
b0f92a668d24a3015692b0869e2b7590a62a380c - rna_seq_reads_2.fq.gz
72b24007759c0086a316d13838021c2571853a16 - rna_seq_reads_3.fq.gz
d4e065787eab480e9cbd2bac6988bc7717464c83 - rna_seq_reads_1.fq.gz
```

5.7 Finding out the size of an item in a dataset

To find the size of a specific item in a dataset one can use the `dtool item properties` command. The command below accesses the properties of the item with the identifier `58f50508c42a56919376132e36b693e9815dbd0c`.

```
$ dtool item properties ~/my_datasets/drone-images␣
↪58f50508c42a56919376132e36b693e9815dbd0c
{
  "relpath": "IMG_8585.JPG",
  "size_in_bytes": 2716446,
  "utc_timestamp": 1505818439.0,
  "hash": "dbcb0d6f22ec660fa4ac33b3d74556f3"
}
```

5.8 Accessing the content of an item in a dataset

When all files are on local disk getting access to them is trivial. However, when files are located in some object storage system in the cloud, access may be less trivial.

dtool solves this problem by providing a call to a method that returns an absolute path on local disk with a promise that the file requested will be available from there when the call returns the path.

The dtool command line interface makes this call available as the command `dtool item fetch`.

Below is an example of this command being used on a local disk file storage.

```
$ dtool item fetch ~/my_datasets/drone-images 58f50508c42a56919376132e36b693e9815dbd0c
/Users/olssont/my_datasets/drone-images/data/IMG_8585.JPG
```

Below is an example of this command being used on a dataset in the S3 bucket `dtool-demo`.

```
$ dtool item fetch s3://dtool-demo/1e47c076-2eb0-43b2-b219-fc7d419f1f16␣
↪3dce23b901709a24cfbb974b70c1ef132af10a67
/Users/olssont/.cache/dtool/s3/1e47c076-2eb0-43b2-b219-fc7d419f1f16/
↪3dce23b901709a24cfbb974b70c1ef132af10a67.txt
```

(continues on next page)

(continued from previous page)

5.9 Processing all the items in a dataset

By combining the use of `dtool identifiers` and `dtool item fetch` it is possible to create basic Bash scripts to process all the items in a dataset.

```
$ DS_URI=~/my_datasets/my_rnaseq_data
$ for ITEM_ID in `dtool identifiers $DS_URI`;
> do ITEM_FPATH=`dtool item fetch $DS_URI $ITEM_ID`;
> echo $ITEM_FPATH;
> done
/Users/olssont/my_datasets/my_rnaseq_data/data/rna_seq_reads_2.fq.gz
/Users/olssont/my_datasets/my_rnaseq_data/data/rna_seq_reads_3.fq.gz
/Users/olssont/my_datasets/my_rnaseq_data/data/rna_seq_reads_1.fq.gz
```

CHAPTER 6

Configuring user name and email

When running the `dtool interactive readme` the default name and email address are `Your Name` and `you@example.com`.

```
$ dtool readme interactive my_dataset
description [Dataset description]:
project [Project name]:
confidential [False]:
personally_identifiable_information [False]:
name [Your Name]:
email [you@example.com]:
username [olssont]:
creation_date [2017-12-14]:
```

These defaults can be configuring the user name and email address.

```
$ dtool config user name "Care A. Bout-Data"
Care A. Bout-Data
$ dtool config user email researcher@famous.uni.ac.uk
researcher@famous.uni.ac.uk
```

Rerunning the previous `dtool readme interactive` command now gives updated defaults when prompting for input.

```
$ dtool readme interactive my_dataset
description [Dataset description]:
project [Project name]:
confidential [False]:
personally_identifiable_information [False]:
name [Care A. Bout-Data]:
email [researcher@famous.uni.ac.uk]:
username [olssont]:
creation_date [2017-12-14]:
```

CHAPTER 7

Configuring the dtool cache directory

When fetching a dataset item from a dataset stored in object storage the file get stored in a cache directory. The default cache directory is:

```
~/.cache/dtool
```

You may want to configure this cache to be in a different location. This can be achieved using the `dtool config cache` command:

```
$ mkdir /tmp/dtool
$ dtool config cache /tmp/dtool
```

Warning: There is no automatic mechanism built into dtool to clear up the cache. It can therefore grow very large if you are working with lots of datasets in object storage.

CHAPTER 8

Configuring a custom README template

When running the `dtool interactive readme` command one is prompted to enter the default descriptive metadata shown below.

```
$ dtool readme interactive my_dataset
description [Dataset description]:
project [Project name]:
confidential [False]:
personally_identifiable_information [False]:
name [Your Name]:
email [you@example.com]:
username [olssont]:
creation_date [2017-12-14]:
```

It is possible to configure the required metadata prompted for by the `dtool readme interactive` command. This requires the creation of a README file making use of the YAML file format.

The default template is shown below.

```
---
description: Dataset description
project: Project name
confidential: False
personally_identifiable_information: False
owners:
  - name: {DTOOL_USER_FULL_NAME}
    email: {DTOOL_USER_EMAIL}
    username: {username}
creation_date: {date}
# links:
#  - http://doi.dx.org/your_doi
#  - http://github.com/your_code_repository
# budget_codes:
#  - E.g. CCBS1H10S
```

To create a custom template that also prompted for a species definition one could create the file `~/custom_dtool_readme.yml` with the content below.

```
---
description: Dataset description
project: Project name
species: A. thaliana
confidential: False
personally_identifiable_information: False
owners:
  - name: {DTOOL_USER_FULL_NAME}
    email: {DTOOL_USER_EMAIL}
    username: {username}
creation_date: {date}
```

To configure the dtool to make use of this template one can use the `dtool config readme-template` command:

```
$ dtool config readme-template ~/custom_dtool_readme.yml
```

The `dtool config readme-template` command sets the `DTOOL_README_TEMPLATE_FPATH` key in the `~/.config/dtool/dtool.json` file. Alternatively one can make use of the `DTOOL_README_TEMPLATE_FPATH` environment variable:

```
$ export DTOOL_README_TEMPLATE_FPATH=~/custom_dtool_readme.yml
```

Re-running the previous `dtool readme interacitve` command now includes a prompt for the species and the default value `A. thaliana`:

```
$ dtool readme interactive my_dataset
description [Dataset description]:
project [Project name]:
species [A. thaliana]:
confidential [False]:
personally_identifiable_information [False]:
name [Your Name]:
email [you@example.com]:
username [olssont]:
creation_date [2017-12-14]:
```

CHAPTER 9

Configuring storage brokers

Some remote storage brokers require extra configuration to enable authentication.

The command below configures access to a Azure storage container named `jicinformatics`:

```
$ dtool config azure set jicinformatics the-secret-token
the-secret-token
```

For information on other storage brokers have a look at their documentation and/or use `dtool config --help` to get more information.

CHAPTER 10

Publishing a dataset

It is possible to publish a datasets hosted in AWS S3 and Microsoft Azure Storage. A dataset is published by making it accessible via the HTTP(S) protocol.

> **Warning:** A published dataset is accessible by anyone in the world with an internet connection!

```
$ dtool publish s3://dtool-demo/ba92a5fa-d3b4-4f10-bcb9-947f62e652db
Dataset accessible at https://dtool-demo.s3.amazonaws.com/ba92a5fa-d3b4-4f10-bcb9-
↪947f62e652db
```

The URL retuned by the `dtool publish` command can be used to interact with the dataset.

```
$ dtool summary https://dtool-demo.s3.amazonaws.com/ba92a5fa-d3b4-4f10-bcb9-
↪947f62e652db
name: hypocotyl3
uuid: ba92a5fa-d3b4-4f10-bcb9-947f62e652db
creator_username: olssont
number_of_items: 339
size: 86.7MiB
frozen_at: 2018-09-12
```

CHAPTER 11

Python API

The `dtool` command line tool is built using the Python API in dtoolcore. This API can also be used to create and interact with datasets directly.

Below is an example showing how to load a dataset from a URI and use it to print out a list of all the data item identifiers in the dataset.

```
>>> from dtoolcore import DataSet
>>> dataset = DataSet.from_uri("bgi-sequencing-12345")
>>> for i in dataset.identifiers:
...     print(i)
...
1c10766c4a29536bc648260f456202091e2f57b4
fbcc24bed36128535a263b74b2e138d7cc43e90c
9ca330a84f3dbbdd457a860b5e3c21c917743dd6
3dce23b901709a24cfbb974b70c1ef132af10a67
78e7f1507da598e9f6a02810c1f846cfc24fb8ad
42f43f49b74ef7f901010965aae71170c9fd3ef6
ab069337b0f86cdad899d57e8de63d5b2b680c85
b55ae3fbe6081eb2ed4ed2c4ea316dbeb943ea2c
```

More information on how to make use of the Python API can be found in the dtoolcore documentation.

CHAPTER 12

Creating plugins

It is possible to create plugins to the `dtool` command line tool. There are two different types of plugins: command line tools and backend storage brokers. The former allows a developer to add custom extensions to the `dtool` command. The latter allows a developer to create an interface for talking to a new type of storage. One could for example create a storage broker to interface with Amazon S3 object storage.

12.1 Extending the `dtool` command line tool

Information on how to extend the `dtool` command line tool is available in the README file of dtool-cli.

Concrete examples making use of this plugin system are:

- dtool-create
- dtool-info

12.2 Creating an interface to a new type of storage

Below are the steps required to create a storage broker for allowing `dtool` to interact with a new backend. A concrete example making use of this plugin system is dtool-irods.

1. Examine the code in `dtoolcore.storagebroker.DiskStorageBroker`.
2. Create a Python class for your storage, e.g. `MyStorageBroker`
3. Add a `MyStorageBroker.key`` attribute to the class, this key is used to lookup an appropriate storage broker when interacting with a dataset
4. Add a `dtoolcore.FileHasher` instance that matches the hashing algorithm used by your storage to your `MyStorageBroker.hasher` attribute
5. Add implementations for all the public functions in `dtoolcore.storagebroker.DiskStorageBroker` class to `MyStorageBroker`

6. Expose the `MyStorageBroker` class as a `dtool.storage_broker` entrypoint, e.g. add a section along the lines of the below to the `setup.py` file:

```
entry_points={
    "dtool.storage_brokers": [
        "MyStorageBroker=my_dtool_storage_plugin:MyStorageBroker",
    ],
},
```

CHAPTER 13

Citing dtool

Olsson TSG, Hartley M. 2019. Lightweight data management with dtool. PeerJ 7:e6562 https://doi.org/10.7717/peerj.6562

CHAPTER 14

CHANGELOG

This project uses semantic versioning. This change log uses principles from keep a changelog.

14.1 [Unreleased]

14.1.1 Added

14.1.2 Changed

14.1.3 Deprecated

14.1.4 Removed

14.1.5 Fixed

14.1.6 Security

14.2 [3.15.0] - 2019-04-26

14.2.1 Added

- `dtool config readme-template` CLI command for configuring the path to a custom readme template
- `dtoolcore._BaseDataSet.base_uri` property
- `dtoolcore.storagebroker.BaseStorageBroker.generate_base_uri` method
- `dtoolcore.utils.DEFAULT_CACHE_PATH` global helper variable
- `dtoolcore.utils.get_config_value_from_file` helper function

- `dtoolcore.utils.write_config_value_to_file` helper function

14.2.2 Changed

- `dtool config cache` now works with one unified cache directory for all storage brokers
- Started using unified environment variable to specify the cache directory `DTOOL_CACHE_DIRECTORY`
- Default cache directory changed set to `~/.cache/dtool`

14.2.3 Fixed

- Fixed defect when username was supplied as two separate strings to `dtool config user name` in CLI

14.3 [3.14.1] - 2018-12-12

14.3.1 Fixed

- Fixed the `dtool config azure set` help text

14.4 [3.14.0] - 2018-11-21

14.4.1 Added

- Added `dtool publish` command
- Added `-f/--format` option to `dtool summary` command to enable output in JSON format
- Added sorting of CSV/TSV/HTML inventories by dataset name

14.4.2 Changed

- Changed default output of `dtool summary` to be human readable YAML

14.5 [3.13.0] - 2018-11-13

14.5.1 Added

- Added support for Windows! :)
- Added `dtool config` command

14.6 [3.12.0] - 2018-09-25

14.6.1 Added

- Added `dtool uuid` command
- Added `dtool item relpath` command

14.7 [3.11.0] - 2018-09-20

14.7.1 Added

- `dtool cp` to replace `dtool copy`
- `dtool readme write` to write readme from file or stdin
- `dtool item overlay` command

14.7.2 Deprecated

- `dtool copy` in favour of `dtool cp`

14.7.3 Removed

- Removed `created_at` field from default README template

14.7.4 Fixed

- Defect in `dtool create` when providing a relative path to the `--symlink-path` option
- Python 2 defect in dealing with unicode in README.yml file when using `dtool readme edit`

14.8 [3.10.0] - 2018-09-11

14.8.1 Added

- `dtoolcore.filehasher.hashsum_digest` helper function
- `dtoolcore.filehasher.md5sum_digest` helper function

14.8.2 Changed

- Improved name from `dtoolcore.filehasher.hashsum` to `dtoolcore.filehasher.hashsum_hexdigest`

14.8.3 Fixed

- Deal with issue in how ruamel.yaml deals with float values

14.9 [3.9.0] - 2018-08-03

14.9.1 Added

- Added ability to update the name of a frozen dataset from the `dtool` CLI
- Added `update_name` method to `DataSet` class (previously only available on `ProtoDataSet` class)

14.10 [3.8.0] - 2018-07-31

Dataset name validation.

14.10.1 Added

- `dtoolcore.generate_admin_metadata` function raises `dtoolcore.DtoolCoreInvalidNameError` if invalid name is provided
- `dtoolcore.utils.name_is_valid` utility function for checking sanity of dataset names
- Validation of dataset name upon creation using dtool CLI
- Validation of dataset name when updating it using dtool CLI

14.10.2 Fixed

- Fixed defect where `dtool ls -q` was listing dataset names rather than URIs making it impossible to process datasets in a BASE_URI programatically
- Make `SymlinkStorageBroker` compatible with dtoolcore 3.4.0

14.11 [3.7.0] - 2018-07-26

Storage broker base class redesign and refactoring.

14.11.1 Added

- Ability to update descriptive metadata in README of frozen datasets
- Validation that the descriptive metadata provided by the `dtool readme edit` command is valid YAML
- Added `dtoolcore.storagebroker.BaseStorageBroker`
- Added logging to the reusable `BaseStorageBroker` methods
- `get_text` new method on `BaseStorageBroker` class
- `put_text` new method on `BaseStorageBroker` class

- `get_admin_metadata_key` new method on `BaseStorageBroker` class
- `get_readme_key` new method on `BaseStorageBroker` class
- `get_manifest_key` new method on `BaseStorageBroker` class
- `get_overlay_key` new method on `BaseStorageBroker` class
- `get_structure_key` new method on `BaseStorageBroker` class
- `get_dtool_readme_key` new method on `BaseStorageBroker` class
- `get_size_in_bytes` new method on `BaseStorageBroker` class
- `get_utc_timestamp` new method on `BaseStorageBroker` class
- `get_hash` new method on `BaseStorageBroker` class
- `get_relpath` new method on `BaseStorageBroker` class
- `update_readme` new method on `BaseStorageBroker` class
- **`DataSet.put_readme` method that can be used to update descriptive metadata** in (frozen) dataset README whilst keeping a copy of the historical README content
- Add `storage_broker_version` key to structure parameters

14.11.2 Fixed

- Stop `copy_resume` function calculating hashes unnecessarily
- Fixed the documentation of the `dtool verify` command

14.12 [3.6.2] - 2018-07-10

14.12.1 Fixed

- Default config file now set in `dtoolcore.utils.get_config_value` if not provided in caller

14.13 [3.6.1] - 2018-07-09

14.13.1 Fixed

- Made download to DTOOL_HTTP_CACHE_DIRECTORY more robust
- Added ability to deal with redirects to enable working with shortened URLs

14.14 [3.6.0] - 2018-07-05

14.14.1 Added

- Bundling of `dtool-http` package

14.14.2 Removed

- Bundling of `dtool-irods` package
- Bundling of `dtool-s3` package

14.15 [3.5.0] - 2018-06-06

14.15.1 Added

- Pre-checks to 'dtool freeze' command to ensure that there is no rogue content in the base of disk datasets
- Added rogue content validation check to DiskStorageBroker.pre_freeze hook

14.16 [3.4.0] - 2018-05-24

14.16.1 Added

- Pre-checks to 'dtool freeze' command to ensure that the item handles are sane, i.e. that they do not contain newline characters
- Pre-checks to 'dtool freeze' command to ensure that there are not too many items in the proto dataset, default to less than 10000

14.17 [3.3.1] - 2018-05-18

14.17.1 Fixed

- Defect where inventory html template is not included in Python package on PyPi

14.18 [3.3.0] - 2018-05-18

14.18.1 Added

- Add "created_at" key to the administrative metadata
- `dtool inventory` command for generating csv/tsv/html inventories of collections of datasets
- Added support for `-h` flag as well as `--help`
- Added timestamp to logging output

14.18.2 Fixed

- Improved handling of URIs in validation code
- Fixed defect where running `dtool item properties` with an invalid identifier resulted in a KeyError exception being propagated to the user

- Fixed defect where `dtool verify` did not compare file sizes
- Fixed timestamp defect in DiskStoragBroker

14.19 [3.2.1] - 2018-05-01

14.19.1 Fixed

- Fixed issue arising from a file being put into iRODS and the connection breaking before the appropriate metadata could be set on the file in iRODS. See also: https://github.com/jic-dtool/dtool-irods/issues/7

14.20 [3.2.0] - 2018-02-09

Release to make it easier to create symlink datasets in an automated fashion.

14.20.1 Changed

- Simplified the way to specify the symbolic link path in the SymLinkStorageBroker
- The path to the data when creating a symlink dataset is now specified using the `-s/--symlink-path` option rather than being something that is prompted for. This makes it easier to create symlink datasets in an automated fashion.

14.21 [3.1.0] - 2018-02-05

14.21.1 Added

- `--resume` option to `dtool copy` command
- `--quite` and `--verbose` options to `dtool ls` and improved formatting
- Add `dtoolcore.copy_resume` function

14.22 [3.0.0] - 2018-01-18

This release makes use of the dtoolcore version 3.0.0 API, which improves the handling of URIs and adds more metadata describing the structure of datasets.

Another major feature of this release is the addition of an S3 storage broker that can be used to interact with Amazon's S3 object storage.

14.22.1 Added

- AWS S3 object storage broker
- Writing of `.dtool/structure.json` file to the DiskStorageBroker; a file for describing the structure of the dtool dataset in a computer readable format

- Writing of `.dtool/README.txt` file to the DiskStorageBroker; a file for describing the structure of the dtool dataset in a human readable format
- Writing of `.dtool/structure.json` file to the IrodsStorageBroker; a file for describing the structure of the dtool dataset in a computer readable format
- Writing of `.dtool/README.txt` file to the IrodsStorageBroker; a file for describing the structure of the dtool dataset in a human readable format

14.22.2 Changed

- Make use of dtoolcore version 3 API

14.22.3 Fixed

- Removed the historical `dtool_readme` key/value pair from the administrative metadata (in the .dtool/dtool file)

14.23 [2.4.0] - 2017-12-14

14.23.1 Added

- Ability to specify a custom README.yml template file path.
- Ability to configure the full user name for the README.yml template using `DTOOL_USER_FULL_NAME`

14.23.2 Fixed

- Made `.dtool/manifest.json` content created by DiskStorageBroker human readable by adding new lines and indentation to the JSON formatting.
- Made the DiskStorageBroker.list_overlay_names method more robust. It no longer falls over if the `.dtool/overlays` directory has been lost, i.e. by cloning a dataset with no overlays from a Git repository.
- Fixed defect where an incorrect URI would get set on the dataset when using `DataSet.from_path` class method on a relative path
- Made the YAML output more pretty by adding more indentation.
- Replaced hardcoded `nbi.ac.uk` email with configurable `DTOOL_USER_EMAIL` in the default README.yml template.
- Fixed `IrodsStorageBroker.generate_uri` class method
- Made `.dtool/manifest.json` content created by IrodsStorageBroker human readable by adding new lines and indentation to the JSON formatting.
- Added rule to catch `CAT_INVALID_USER` string for giving a more informative error message when iRODS authentication times out

14.24 [2.3.2] - 2017-10-25

14.24.1 Fixed

- Fixed issue where the symbolic link was not fully resolved when creating a symlink dataset that used the terminal to prompt for the data directory

14.25 [2.3.1] - 2017-10-25

14.25.1 Fixed

- More graceful exit if one presses Cancel in file browser when creating a symlink dataset
- Data directory now falls back on click command line prompt if TkInter has issues when creating a symlink dataset

14.26 [2.3.0] - 2017-10-23

14.26.1 Added

- `pre_freeze_hoook` to the stroage broker interface called at the beginning of `ProtoDataSet.freeze` method.
- `--quiet` flag to `dtool create` command
- `dtool overlay ls` command to list the overlays in dataset
- `dtool overlay show` command to show the content of a specific overlay

14.26.2 Changed

- Improved speed of freezing a dataset in iRODS by making use of caches to reduce the number of calls made to iRODS during this process
- `dtool copy` now specifies target location using URI rather than using the `--prefix` and `--storage` arguments

14.26.3 Fixed

- Made the `DiskStorageBroker.create_structure` method more robust
- More informative error message when iRODS has not been configured
- More informative error message when iRODS authentication times out
- Stopped client hanging when iRODS authentication has timed out
- storagebroker's `put_item` method now returns relpath
- Made the `IrodsStorageBroker.create_structure` method more robust by checking if the parent collection exists
- Made error handling in `dtool create` more specific

- Added propagation of original error message when `StorageBrokerOSError` captures in `dtool create`

14.27 [2.2.0] - 2017-10-09

14.27.1 Added

- `dtool ls` can now be used to list the relpaths of the items in a dataset
- `-f/--full` flag to `dtool diff` command to include checking of file hashes
- `-f/--full` flag to `dtool verify` command to include checking of file hashes

14.27.2 Changed

- `dtool ls` now works with URIs rather than with prefix and storage arguments
- `dtool diff` now only compares identifiers and file sizes by default
- `dtool verify` now only compares identifiers and file sizes by default

14.27.3 Fixed

- Made `DiskStorageBroker.list_dataset_uris` class method more robust

14.28 [2.1.2] - 2017-10-05

14.28.1 Fixed

- Set the correct dependency to actually get fix reported in 2.1.1

14.29 [2.1.1] - 2017-10-05

14.29.1 Fixed

- Fixed defect in iRODS storage broker where files with white space resulted in broken identifiers

14.30 [2.1.0] - 2017-10-04

14.30.1 Added

- `dtool readme show` command that returns the readme content
- `--quiet` flag to `dtool copy` command

14.30.2 Changed

- Improved the `dtool readme --help` output

14.30.3 Fixed

- Progress bar now shows information on individual items being processed
- `dtool ls` now works with relative paths
- Fix defect where `IrodsStorageBroker.put_item` raised SystemError when trying to overwrite an existing file

14.31 [2.0.2] - 2017-09-25

14.31.1 Fixed

- Better validation of input in terms of base vs proto vs frozen dataset URIs
- Fixed bug where copy creates an intermediate proto dataset that self identifies as a frozen dataset.
- Fixed potential bug where a copy could convert a proto dataset to a dataset before all its overlays had been copied over
- Fixed type of "frozen_at" time stamp in admin metadata: from string to float

14.32 [2.0.1] - 2017-09-20

14.32.1 Fixed

- Made version requirements of dtool sub-packages explicit

14.33 [2.0.0] - 2017-09-14

Initial release of `dtool` as a meta package.

CHAPTER 15

MIT License