Datalad-OSF

Release 0.3.0+15.g52545e3.dirty

DataLad team

Dec 08, 2023

CONTENTS

This DataLad extension enables DataLad to work with the Open Science Framework (OSF). Use it to publish your dataset's data to an OSF project to utilize the OSF for dataset data storage and share and collaborate on datasets.

The extension was created during the OHBM Hackathon 2020 and wouldn't have been possible without a dedicated team of volunteers. If you want to get in touch or on board as well, please see our *contributing guidelines*.

CHAPTER

ONE

DOCUMENTATION OVERVIEW

1.1 Introduction

1.1.1 The Open Science Framework

The Open Science Framework (OSF), developed and maintained by the Center for Open Science (COS), is a tool that promotes open, centralized workflows by enabling capture of different aspects and products of the research life-cycle, including developing a research idea, designing a study, storing and analyzing collected data, and writing and publishing reports or papers. In the scientific community, it is commonly used for registered reports, as a preprint server, and for study archival and data sharing. In order to use the OSF, a free registration is required.

The core functionality of the OSF is its ability to create and develop *projects*, with a project being a private or public workspace for collaboration, data sharing, or data archival. Projects have an associated storage (either via OSF storage or third party providers) for (large) data, and can have *components*, associated sub-projects. Each OSF user, project, component, and file is given a unique, persistent uniform resource locator (URL) to enable sharing and promote attribution. Projects can also be assigned digital object identifiers (DOIs) and archival resource keys (ARKs) if they are made publicly available.

The OSF is a powerful, accessible, and free data sharing and collaboration platform for researchers. However, the storage that OSF provides is not unlimited. Please check their current storage capacity and check if it fits your needs.

1.1.2 Goal of the extension

This extension allows DataLad to work with the Open Science Framework (OSF) to make sharing and collaboration on data or DataLad datasets even easier. It comes with several features that enable the following main use cases:

1. Export existing datasets to the OSF
2. Clone published datasets from the OSF
3. Utilize OSF projects as a third party storage provider for annexed data
4. Export single-view snapshots of datasets to an OSF project

To enable these use cases, a dataset is published as an OSF project, and its OSF storage is used as a git-annex special remote to publish (large) file contents. Major OSF flexibility is exposed to control whether the resulting project is private (default) or public, and to attach meaningful metadata to it. You can find out demonstrations of these use cases in the *Tutorial*.

1.1.3 What can I use this extension for?

You can use this extension to publish, store, collaborate on, or share your dataset (data) via the OSF. Here is some inspiration on what you could do:

- Publish your study (including its version history, data, code, results, and provenance) as a DataLad dataset to the OSF. Share the project's OSF URL with colleagues and collaborators to give them easy access to your work with a single `datalad clone`.
- Clone a friend's dataset – from the OSF!
- Use the OSF as a special remote to store data in the annex of your dataset. With this, you can publish a dataset to GitHub or similar Git repository hosting services, and have your data published to the OSF (via a publication dependency). Your dataset will be exposed and available on GitHub, while data is stored on and retrieved from the OSF.
- Take a version snapshot of all of your dataset's files and export them to the OSF. This publishes one version of your project in a human-readable fashion to the OSF to make it available to the outside world.

`datalad-osf` comes with a range of hidden convenience functions for OSF interactions. Importantly, you will not need to create OSF projects via the OSF web interface – given appropriate credentials, `datalad create-sibling-osf` will create new projects under your user account and report back the generated URL.

1.1.4 What can I not use this extension for?

- This tool does not work for data that is stored in a storage service other than the OSF, and within the OSF, only OSF storage, no other third party storage, is supported. Please refer to the list of special remotes as hosted by the git-annex website for other storage services and how to use them with DataLad.
- Also, be mindful that OSF storage imposes a maximum file size of 5GB. Individual files larger than 5GB can not be published with this extension.
- Finally, the starting point for working with this extension is a (published) DataLad dataset, not a regular OSF project. This extension will not transform normal OSF projects into datasets, but expose DataLad datasets as OSF projects.

1.2 Quickstart

1.2.1 Requirements

DataLad and `datalad-osf` are available for all major operating systems (Linux, MacOS, Windows 10[1]). The relevant requirements are listed below.

An OSF account
: You need an OSF account to be able to interact with it. If you don't have an account yet, register here – its free!

DataLad
: If you don't have DataLad and its underlying tools (git, git-annex) installed yet, please follow the instructions from the datalad handbook.

[optional] An account on a Git repository hosting site
: You should consider having an account on one or more repository hosting sites such as GitHub , GitLab, Bitbucket or similar.

[1] While installable for Windows 10, the extension may not be able to perform all functionality documented here. Please get in touch if you are familiar with Windows to help us fix bugs.

1.2.2 Installation

`datalad-osf` is a Python package available on pypi and installable via pip.

```
# create and enter a new virtual environment (optional)
$ virtualenv --python=python3 ~/env/dl-osf
$ . ~/env/dl-osf/bin/activate
# install from PyPi
$ pip install datalad-osf.
```

1.2.3 Getting started

Here's the gist of some of this extension's functionality. Checkout the *Tutorial* for more detailed demonstrations.

First, *provide your credentials*:

```
# provide your OSF credentials, ideally as a token:
$ datalad osf-credentials
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token: <your token here>
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token (repeat): <your token here>
osf_credentials(ok): [authenticated as <user> <e-mail>]
```

Next, create a sibling on the OSF for a DataLad dataset of your choice. Choose between different sibling modes to adjust how much of your dataset can be published and how it will be displayed, adjust whether your project should be private or public, attach additional meta data, or configure local sibling properties. The minimal example below will create a new (private) project with minimal metadata on the OSF and apply the necessary configurations to publish your complete dataset to it.

```
# inside of a DataLad dataset
$ datalad create-sibling-osf --title best-study-ever -s osf
create-sibling-osf(ok): https://osf.io/czgpf/
[INFO   ] Configure additional publication dependency on "osf-storage"
configure-sibling(ok): /home/me/mydataset (sibling)
```

Afterwards, publish your dataset to the OSF sibling project to share it or collaborate with others:

```
$ datalad push --to osf
```

Finally, you or others can clone it using its project ID. All annexed data in this dataset will be available via `datalad get`.

```
$ datalad clone osf://czgpf/
```

Curious to find out more? Read on in the *Tutorial* for more functionality and use cases.

HELP! I'm new to this!

If this is your reaction to reading the words DataLad dataset, sibling, or dataset publishing, please head over to the DataLad Handbook for an introduction to DataLad.

1.3 Tutorial

1.3.1 Step 1: Authentication

`datalad-osf` needs to communicate with the OSF to create and modify projects under an associated user account. To enable this, the associated user needs to be authenticated using the **`osf-credentials`** command. Therefore, as the very first step, `datalad osf-credentials` needs to be ran to authenticate a user. Unless credentials expire or change, this command needs to be ran only once per user and system.

Setting credentials

To set credentials, run `datalad osf-credentials` anywhere on your system. This command prompts for user credentials and stores them in your system's secure credential store for OSF operations.

```
# the default authentication method is token
$ datalad osf-credentials
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token: <your token here>
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token (repeat): <your token here>
osf_credentials(ok): [authenticated as <user> <e-mail>]
```

Two different methods of authentication are supported and can be set with the `--method` flag:

- `token`: A personal access token. This is the recommended authentication type and default. Generate a personal access token under your user account at osf.io/settings/tokens. Make sure to create a `full_write` token to be able to create OSF projects and upload data to OSF!
- `userpassword`: Your username and password combination from the OSF web interface.

```
# authenticate with user name and password
$ datalad osf-credentials --method userpassword
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪account provides information on how to gain access
user: <your e-mail address>

password: <your password here>
password (repeat): <your password here>
osf_credentials(ok): [authenticated as <user name>]
```

The credentials are stored within a system's encrypted keyring and DataLad retrieves them automatically for all future interactions with the OSF. Information on which user's credentials are stored can be found by re-running `datalad osf-credentials`.

```
$ datalad osf-credentials
osf_credentials(ok): [authenticated as <user name> <e-mail>]
```

Environment variables

Alternatively, credentials can be set via environment variables: `OSF_TOKEN`, or both `OSF_USERNAME` and `OSF_PASSWORD`, as in

```
export OSF_TOKEN=YOUR_TOKEN_FROM_OSF
```

Resetting credentials

If credentials change they can be re-set using the `--reset` flag:

```
# token method is used by default, use --method userpassword for user + password␣
↪credentials
$ datalad osf-credentials --reset
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token: <your token here>
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token (repeat): <your token here>
osf_credentials(ok): [authenticated as <user name> <e-mail>]
```

Invalid credentials

If you supply invalid credentials such as a mismatching user name and password combination or a wrong token, you will see the following error:

```
$ osf_credentials(error): None [Invalid credentials]
```

Please check for spelling mistakes, check your user name and password combination under your user account, or re-generate a token, and reset your credentials to fix this.

1.3.2 Step 2: Create an OSF sibling

Once authenticated, DataLad can – if called from within a DataLad dataset – create and modify projects on the OSF and publish annexed data, a single version view, or the complete dataset to it. The command that enables this is **`datalad create-sibling-osf`**. It supports different modes, exposes a large number of features from the OSF web interface and yields a custom dataset configuration for your use case at hand. This section introduces the command and its functionality, and the upcoming use cases demonstrate several workflow types it can be used for.

What's a sibling?

A sibling is a dataset clone that a given DataLad dataset knows about. In most cases, changes can be retrieved and pushed between a dataset and its sibling. It is the equivalent of a *remote* in Git. The **`datalad create-sibling-osf`** command can create a dataset clone under an authenticated user account on the OSF as a new project.

General command

When relying on default parameters, `create-sibling-osf` requires only a project name for the resulting OSF project (`--title`) and a sibling name (`-s`/`--name`, which defaults to `osf`).

```
# within a dataset mydataset
$ datalad create-sibling-osf --title <my-new-osf-project-title> -s <my-sibling-name>
create-sibling-osf(ok): https://osf.io/czgpf/
[INFO   ] Configure additional publication dependency on "<my-sibling-name>-storage"
configure-sibling(ok): <path/to/mydataset> (sibling)
```

In the default modes of operation and in most other modes, this command will create one project on the OSF (reported under its URL in the command summary) and two dataset siblings: One sibling to publish Git history and files stored in Git to, and a *storage* sibling to which annexed data will be published.

---

dataset publishing

Note that data still needs to be *published* to be available on the OSF after sibling creation. The relevant commands for this are dependent of the sibling *mode*. If used in the default mode, both siblings will be automatically configured such that a single `datalad push` is sufficient to publish the complete dataset. For general information on publishing datasets, please refer to this Handbook chapter, and for more information on publishing depending on sibling mode, please pay close attention to the paragraph *Sibling modes* and the upcoming use cases.

Sibling configuration

`create-sibling-osf` has a number of parameters that expose the underlying flexibility of DataLad, git-annex, and the OSF. The most important one is the *mode* (`--mode`) parameter. Depending on the mode for your OSF sibling, you will be able to publish different aspects of your dataset to the OSF, and each mode requires different commands for publishing. Other important parameters include the `--public` flag (for access control), and `--tag`, `--category` and `--description` for additional project meta data. For a complete overview of all parameters, see the *command documentation*.

Sibling modes

`create-sibling-osf` supports several modes that determine the functionality and usage of the resulting sibling.

- `annex` (default): **You can publish the complete dataset to the resulting OSF project**. This includes all Git history and annexed data. Afterwards, the OSF project URL can be cloned to retrieve the dataset, and `datalad get` will be able retrieve all file contents, even older versions. This mode is the most convenient if you aim to share complete datasets with all data and version history. Note that the dataset representation in the OSF project is not as readable as in a local dataset (clone), but a non-human readable representation[1] tuned to enable cloning. Publishing the dataset requires only `datalad push`.

[1] What exactly is a non-human readable representation? On the OSF, the Git history will be compressed to a text file and a zip file, while all annexed data will appear under the hash it is stored in in the git-annex object tree, i.e., with an obscured file name. If you are interested in finding out more about this, take a look at this section in the DataLad handbook. Once cloned from the OSF to a local file system, the dataset will have its usual, human-readable format.

- `export`: **You can push the Git history of a dataset as well as one snapshot of its data to the resulting OSF project**. Afterwards, the OSF project URL can be cloned to retrieve the dataset and `datalad get` will be able to retrieve all file contents *in one version*. Compared to the `annex` mode, the dataset representation on the OSF is human-readable, but only one version of each file can be published. This mode is convenient if you want to share a dataset and its history in a human-readable way but only make one version of it available. Publishing Git history requires `git push` or `datalad push`, and exporting a single view of the data must be done via `git-annex export`.
- `gitonly`: **You can push the Git history of a dataset, but no annexed data to the resulting OSF project**. Afterwards, the OSF project URL can be cloned to retrieve the dataset, but `datalad get` will not be able to retrieve file contents. This can be convenient if you want to use the OSF as an alternative to GitHub. Note that the representation of the dataset is not human-readable, but tuned for cloning. Publishing Git history requires `git push` or `datalad push`.
- `exportonly`: **You can export the dataset in a human-readable way in one version**. Note that this type of sibling can not be cloned from the OSF. This option is the most convenient if you want to make one snapshot of your dataset available via the OSF. Exporting needs to be done via `git-annex export` and your dataset will only get a storage sibling.

In deciding which mode suits your use case you can consider the following questions:

1. Do you want collaborators to be able to `datalad clone` your project? If yes, go for `annex`, `export`, or `gitonly`
2. Do you want to share your data? If yes, go for `annex`, or – if you're okay with sharing only a one version per file – `export` and `export only`
3. Do you care how data looks like on the OSF? If not, go for `annex`, if yes, use one of the `export` modes. Find out more about this in the *tutorial on exporting data*.

Access Management: Public or private projects

By default, any new project created with `create-sibling-osf` is a private OSF project that can only be accessed by its creator and collaborators added via OSF's interface. To make a project public, you can either transform it into a public project via the web interface, or use the `--public` flag of `create-sibling-osf` to create it publicly from the very start. This constitutes a convenient access management system for your dataset.

OSF project metadata

Meta data helps to make your project discoverable and understandable. The OSF provides several means of attaching meta data to a project: Tags and Categories. By default, two tags are created upon project creation: "DataLad dataset" and the unique ID of your dataset. Any amount of arbitrary additional tags can be specified with one or more `--tag` options. Note that each tag needs to be specified with its own `--tag` parameter.

The category of a project determines the small icon displayed in a project and helps search organization. You can chose one out of several categories ("analysis", "communication", "data", "hypothesis", "instrumentation", "methods and measures", "procedure", "project", "software", "other") and specify it using the `--category` parameter. By default, the category "data" is used.

1.3.3 Walk-Trough

The upcoming use cases are walk-throughs and meant as code-along tutorials. If you want, open a terminal and code along to try out each method. If you have DataLad and `datalad-osf` installed, each tutorial will not take more than 5 minutes.

As a general preparation, build an example dataset and configure OSF credentials for reuse in all usecases. You can execute all following examples in this dataset.

Create an Example Dataset

For the sake of this tutorial, let's create a small example DataLad dataset.

```
# collab_osf being the name of your new dataset
$ datalad create collab_osf
```

After dataset creation, populate it with some content (just like in the Datalad Handbook):

```
$ cd collab_osf
# add a PDF file to git-annex
$ datalad download-url http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.
↪pdf \
  --dataset . \
  -m "add beginners guide on bash" \
  -O books/bash_guide.pdf

download_url(ok): /tmp/collab_osf/books/bash_guide.pdf (file)
add(ok): /tmp/collab_osf/books/bash_guide.pdf (file)
save(ok): /tmp/collab_osf dataset)
action summary:
 add (ok: 1)
 download_url (ok: 1)
 save (ok: 1)
# add a text file to Git
$ mkdir code
$ echo "This is just an example file just to show the different ways of saving data in a␣
↪DataLad dataset." > code/example.txt
$ datalad save --to-git -m "created an example.txt"
add(ok): /tmp/collab_osf/code/example.txt (file)
save(ok): /tmp/collab_osf(dataset)
action summary:
  add (ok: 1)
  save (ok: 1)
```

Authenticate

First, if you haven't done so yet, configure either your OSF credentials (username and password) or an OSF access token, either as environment variables, or using `datalad osf-credentials`. Below, we use an OSF access token:

```
$ datalad osf-credentials
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token: <your token here>
You need to authenticate with 'https://osf.io' credentials. https://osf.io/settings/
↪tokens provides information on how to gain access
token (repeat): <your token here>
osf_credentials(ok): [authenticated as <user> <e-mail>]
```

More information on authentication is detailed in the section *Step 1: Authentication*.

1.3.4 Use case 1: Publishing and cloning datasets

Problem statement

Imagine you have been creating a reproducible workflow using DataLad from the get go. Everything is finished now, code, data, and paper are ready. Last thing to do: Publish your data, code, results, and workflows – ideally, all together, easily accessible, and also fast.

The solution: Publish the complete dataset to the OSF and let others clone the project to get access to data, code, version history, and workflows. Therefore, you decide on the `annex` sibling mode.

Creating the OSF sibling

Given OSF credentials are set, we can create a sibling in `annex` mode. We will also make the project public (`--public`), and attach some meta data (`--category`, `--tag`) to it.

The code below will create a new public OSF project called `best-study-ever`, a dataset sibling called `osf-annex`, and a readily configured storage sibling `osf-annex-storage`. The project on the OSF will have a description with details on how to clone it and some meta data.

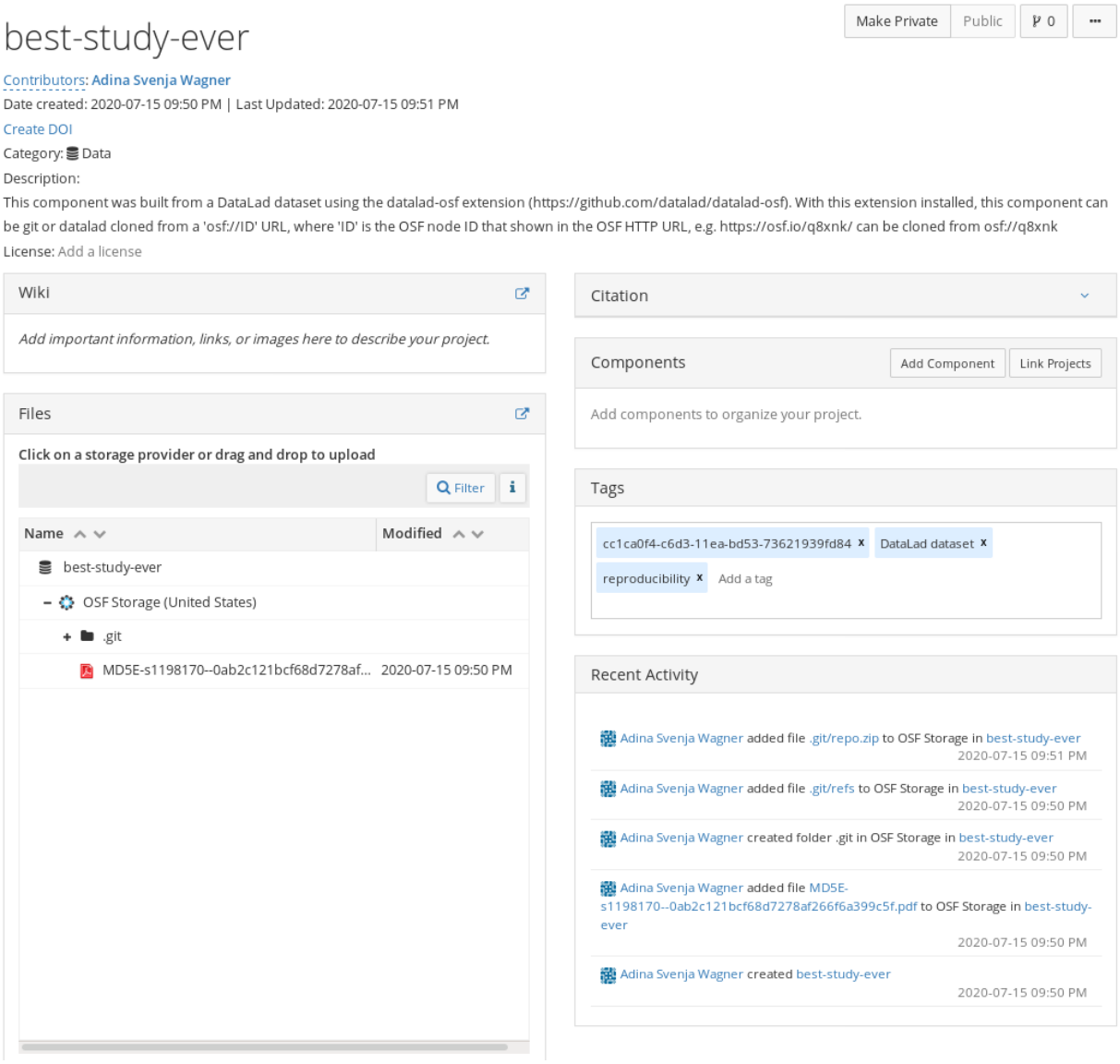
```
# inside of the tutorial DataLad dataset
$ datalad create-sibling-osf --title best-study-ever \
  -s osf-annex \
  --category data \
  --tag reproducibility \
  --public

create-sibling-osf(ok): https://osf.io/<id>/
[INFO   ] Configure additional publication dependency on "osf-annex-storage"
configure-sibling(ok): /tmp/collab_osf (sibling)
```

Publishing the dataset

Afterwards, all that's left to do is a `datalad push` to publish the dataset to the OSF.

```
$ datalad push --to osf-annex
```

The resulting dataset has all data and its Git history, but is not as human-readable as on a local computer:



Cloning the dataset

The dataset can be cloned with an `osf://<id>` URL, where ID is the project ID assigned at project creation:

```
$ datalad clone osf://n6bgd/ best-study-ever
  install(ok): /tmp/best-study-ever (dataset)
```

All data can subsequently be obtained using `datalad get`.

1.3.5 Use case 2: Export a human-readable dataset to OSF

Problem statement

Imagine you have been collecting data and want to share it with others. All your colleagues have only sparse experience with the command line, but are frequent OSF users. Therefore, you place all data in a dataset and export the latest version of your data to the OSF in a human readable way, for others to view it conveniently in the web interface.

As you want a human-readable representation and decide to not share the version history, but only the most recent state of the data, you pick the `exportonly` sibling mode.

Creating the OSF sibling

Given OSF credentials are set, we can create a sibling in `export-only` mode. We will also make the project public (`--public`), and attach a custom description (`--description`) to it.

The code below will create a new public OSF project called `best-data-ever`, a dataset sibling called `osf-exportonly-storage`. This sibling is a sole storage sibling – in technical terms, a git-annex special remote. It will not contain version history, but all your dataset's files.

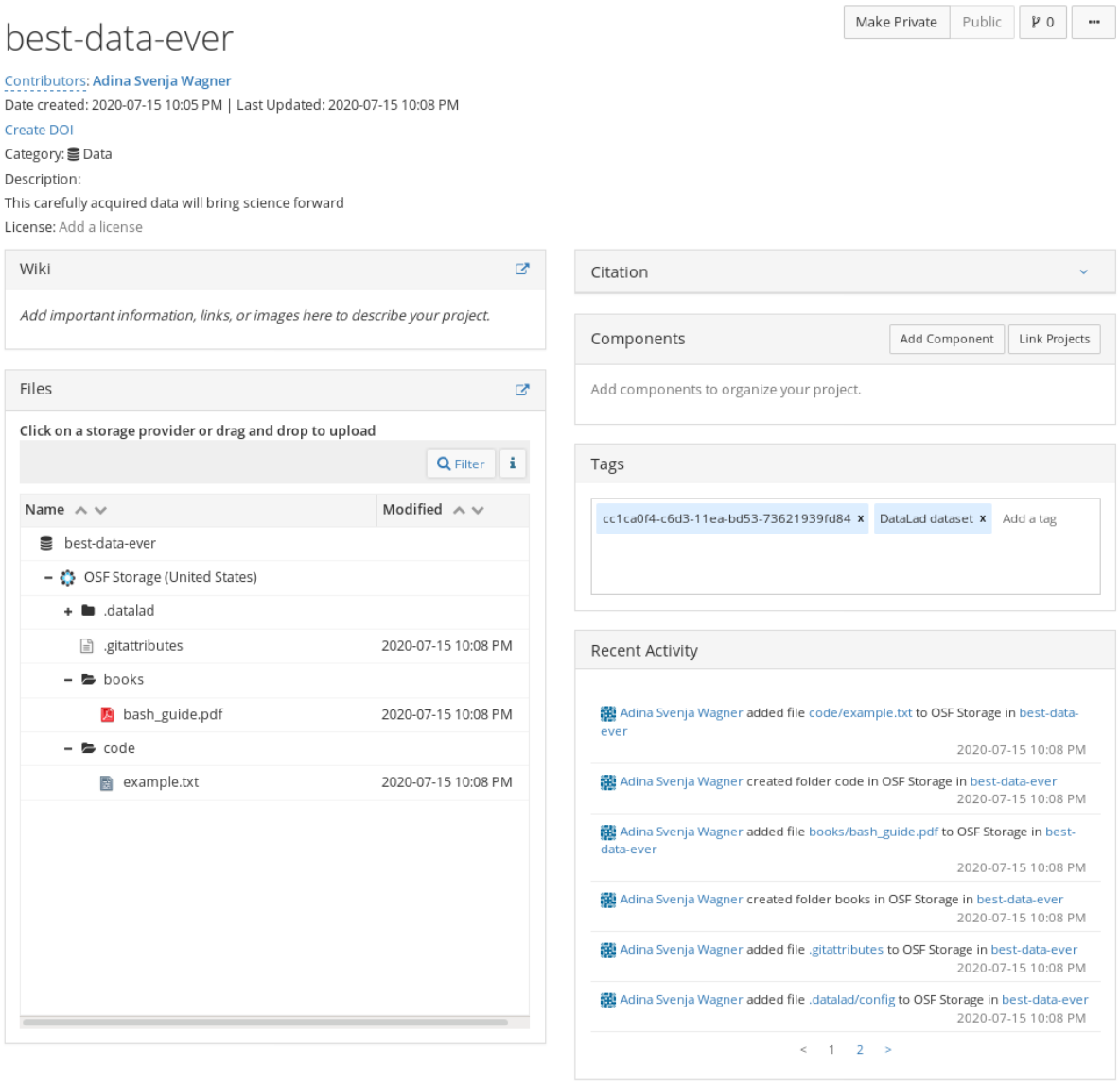
```
# inside of the tutorial DataLad dataset
$ datalad create-sibling-osf --title best-data-ever \
  --mode exportonly \
  -s osf-export \
  --description "This carefully acquired data will bring science forward" \
  --public

create-sibling-osf(ok): https://osf.io/<id>/
# note that the sibling name as an appended "storage" suffix!
$ datalad siblings
 .: here(+) [git]
 .: osf-annex(-) [osf://n6bgd (git)]
 .: osf-export-storage(+) [osf]          # created in this example
 .: osf-annex-storage(+) [osf]
```

Publishing the dataset

To publish the current state (the `HEAD`) of the dataset, simply run:

```
$ datalad push  --to osf-export-storage
  copy(ok): .datalad/.gitattributes
  copy(ok): .datalad/config
  copy(ok): .gitattributes
  copy(ok): books/bash_guide.pdf
  copy(ok): code/example.txt
  action summary:
     copy (ok: 5)
```

The resulting project has a human readable structure, and all its data can be viewed and downloaded via the OSF interface. It is not possible to clone this dataset with DataLad, however potential users can still download it from the standard OSF interface.



1.3.6 Use case 3: Using the OSF as a data store for a GitHub-based project

Problem statement

Imagine you are a PhD student and want to collaborate on a fun little side project with a student at another institute. You agree that your code will be hosted on GitHub due to its easier accessibility and greater feature selection. But what about the data you are collecting? The Dropbox is already full (DataLad third party providers). And Amazon services don't seem to be your best alternative. Suddenly you remember, that you got an OSF account recently: You decide to publish your dataset to GitHub and your data to the OSF and link both via a publication dependency.

Therefore, you go with a sibling in `annex` mode. While others *can* clone it from the OSF, you mostly utilize it for data access. Others clone the dataset from GitHub and are unaware where your data is stored.

Creating the OSF sibling

Given OSF credentials are set, we can create a sibling in `annex` mode.

As in use case 1, the code below will create a new public OSF project called `our-study-data`, a dataset sibling called `osf-annex2`, and a readily configured storage sibling `osf-annex2-storage`.

```
# inside of the tutorial DataLad dataset
$ datalad create-sibling-osf --title our-study-data \
  -s osf-annex2 \
  --category data \
  --tag reproducibility \
  --public

create-sibling-osf(ok): https://osf.io/<id>/
[INFO   ] Configure additional publication dependency on "osf-annex2-storage"
configure-sibling(ok): /tmp/collab_osf (sibling)
```

Creating a sibling on GitHub

As the goal is to use OSF for data storage and expose the dataset also via GitHub, we're not done yet. We can set-up a GitHub Remote with name `github` and include a publication dependency to the OSF storage sibling – that way, when we publish our dataset to GitHub, the data files get automatically uploaded to OSF.

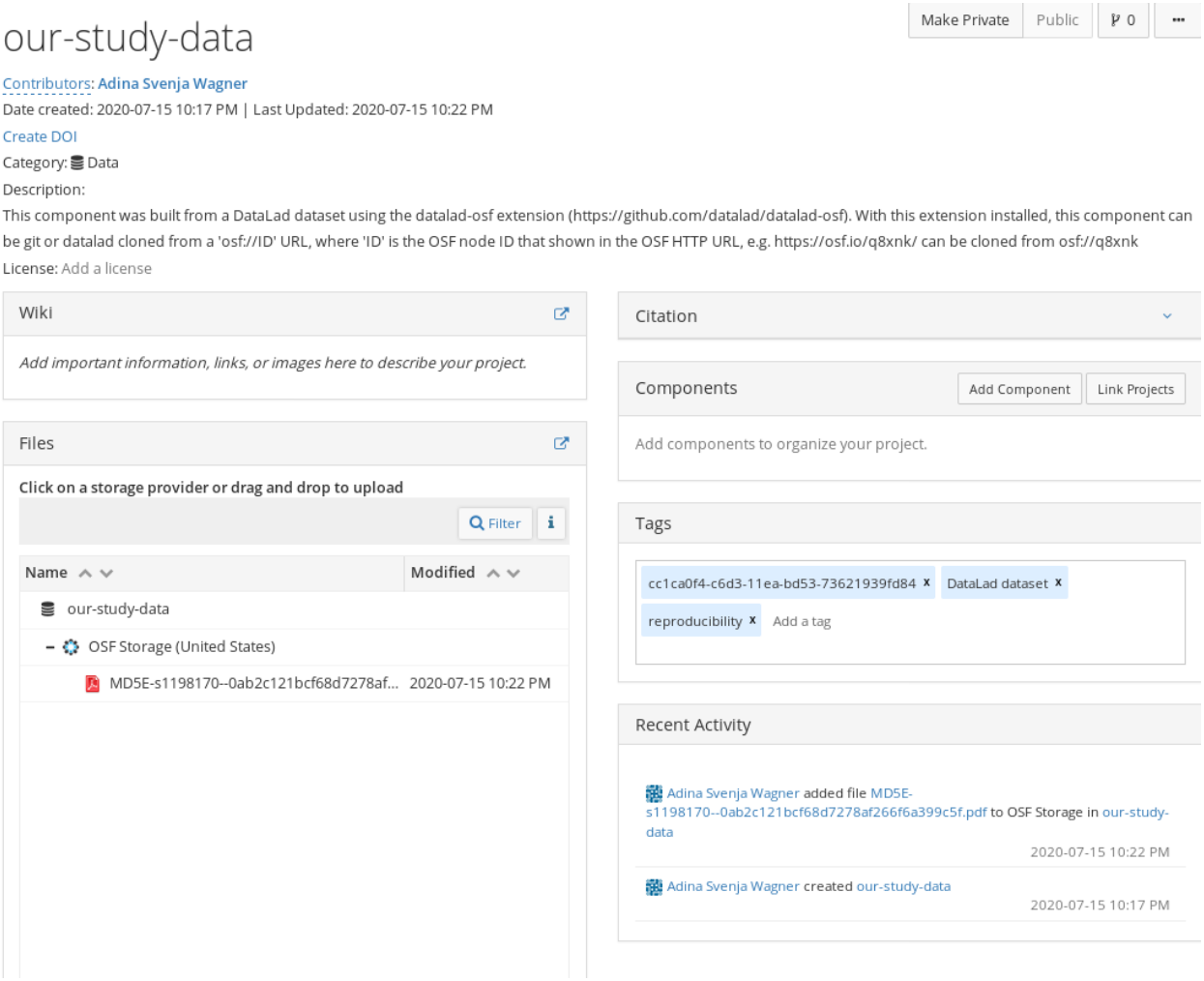
```
$ datalad create-sibling-github our-study-data \
  -s github \
  --github-login LOGIN \
  --publish-depends osf-annex2-storage

You need to authenticate with '<login>@github' credentials. https://github.com/login␣
↪provides information on how to gain access
password: <password>
You need to authenticate with '<login>@github' credentials. https://github.com/login␣
↪provides information on how to gain access
password (repeat): <password>
[INFO   ] Configure additional publication dependency on "osf-annex2-storage"
.: github(-) [https://<login>@github.com/<login>/our-study-data.git (git)]
'https://<login>@github.com/<login>/our-study-data.git' configured as sibling 'github'␣
↪for Dataset(/tmp/collab_osf)
```

Publish the dataset to GitHub and its data to OSF

Because a publication dependency to the OSF is set up, a `datalad push` to GitHub is sufficient.

```
$ datalad push --to github
  Push to 'github':  [...]           | 1.00/4.00 [00:00<00:00, 25.9k Steps/s]
  Password for 'https://adswa@github.com': <password>
  copy(ok): /tmp/collab_osf/books/bash_guide.pdf (file) [to osf-annex2-storage...]
  Push to 'github': [...]           | 1.00/4.00 [00:33<01:41, 33.9s/ Steps]
  Update availability for 'github':  [...] | 3.00/4.00 [00:00<00:00, 60.5k Steps/s]
  Password for 'https://adswa@github.com': <password>
  publish(ok): /tmp/collab_osf (dataset) [refs/heads/master->github:refs/heads/master␣
↪[new branch]]
  Update availability for 'github':  [...] | 3.00/4.00 [00:15<00:05, 5.27s/ Steps]
  Publish(ok): /tmp/collab_osf (dataset) [refs/heads/git-annex->github:refs/heads/git-
↪annex [new branch]]
  Update availability for 'github': [...] | 3.00/4.00 [00:15<00:05, 5.27s/ Steps]
```

Afterwards, the dataset can be cloned from GitHub. For a user, the experience will feel similar to use case 1: After cloning, the files in Git and all dataset history are available, all data stored in the annex is retrieved upon `datalad get`. The file content, though, will be retrieved from the OSF, which now serves as a data store for the GitHub repository.



This way you can let OSF handle your data, but still use GitHub to expose your dataset.

1.3.7 Usecase 4: –mode export, The best of both worlds

By now, we've seen several different modes: The `annex` mode creates a clone-able dataset with version history on the OSF, but it is not human readable. The `exportonly` mode creates a human-readable snapshot on the OSF, but it is not clone-able and has no version history.

The best-of-both-worlds compromise between these modes is the `export` mode: It creates a human readable snapshot on the OSF. This snapshot can be cloned, and it includes the datasets version history. However, only the latest version of annexed data is available.

Creating the OSF sibling

Given OSF credentials are set, we can create a sibling in `export` mode. We will also make the project public (–public), and attach category metadata.

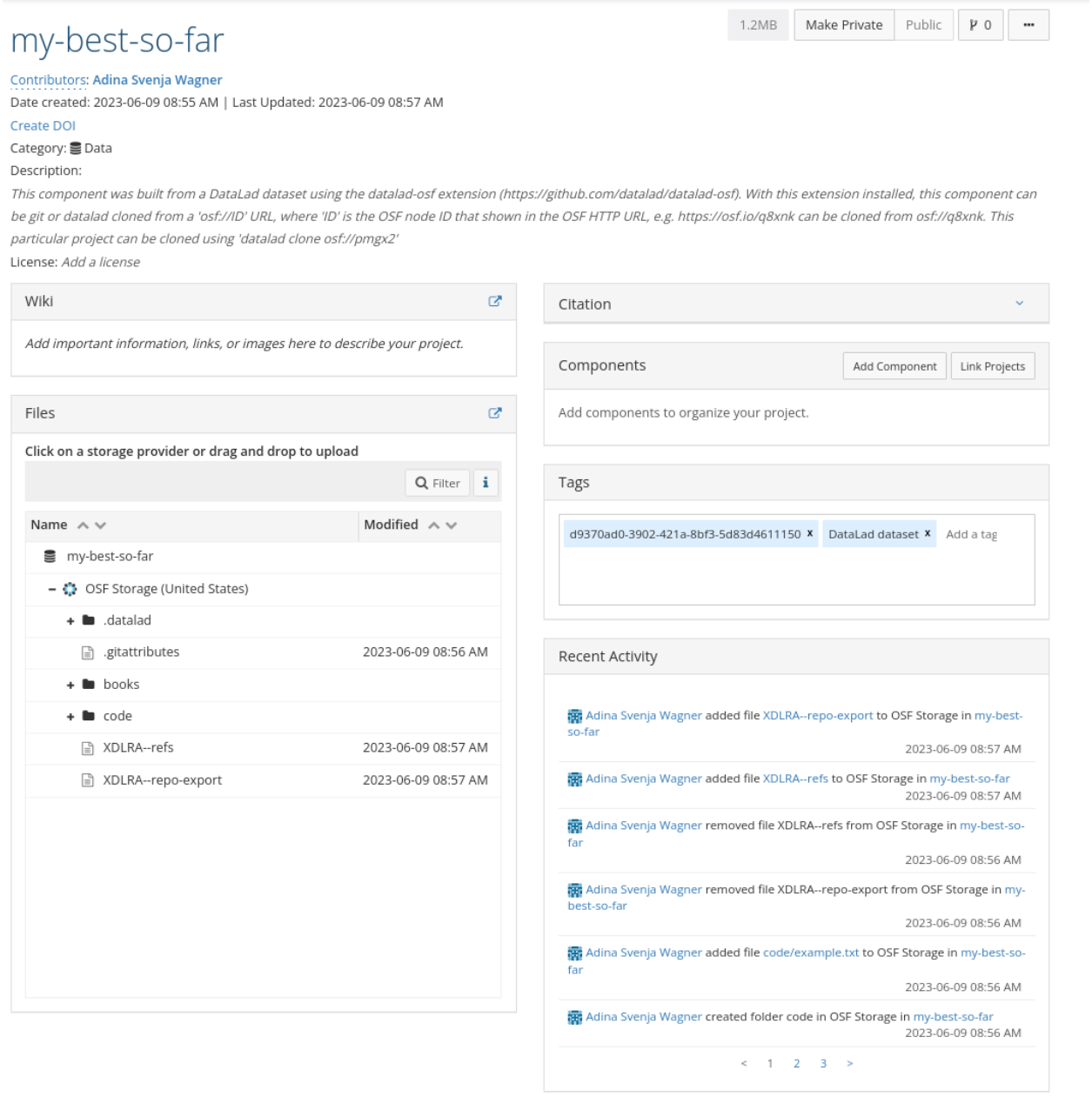
```
$ datalad create-sibling-osf --title my-best-so-far \
  -s osf-export \
  --mode export \
  --public \
  --category data
```

Publishing the dataset

Next, we can push revision history and export a snapshot of all files.

```
$ datalad push --to osf-export
     copy(ok): .datalad/.gitattributes (dataset)
     copy(ok): .datalad/config (dataset)
     copy(ok): .gitattributes (dataset)
     copy(ok): books/bash_guide.pdf (dataset)
     copy(ok): code/example.txt (dataset)
     publish(ok): . (dataset) [refs/heads/master->osf-export:refs/heads/master [new␣
↪branch]]
     publish(ok): . (dataset) [refs/heads/git-annex->osf-export:refs/heads/git-annex␣
↪[new branch]]
 action summary:
   copy (ok: 5)
   publish (ok: 2)
```

And here is how nicely human readable this looks:


my-best-so-far
1.2MB
Make Private
Public
0
Contributors: Adina Svenja Wagner
Date created: 2023-06-09 08:55 AM | Last Updated: 2023-06-09 08:57 AM
Create DOI
Category: Data
Description:
This component was built from a DataLad dataset using the datalad-osf extension (https://github.com/datalad/datalad-osf). With this extension installed, this component can be git or datalad cloned from a 'osf://ID' URL, where 'ID' is the OSF node ID that shown in the OSF HTTP URL, e.g. https://osf.io/q8xnk can be cloned from osf://q8xnk. This particular project can be cloned using 'datalad clone osf://pmgx2'
License: Add a license
Wiki
Add important information, links, or images here to describe your project.
Files
Click on a storage provider or drag and drop to upload
Filter
Name
Modified
my-best-so-far
OSF Storage (United States)
.datalad
.gitattributes
2023-06-09 08:56 AM
books
code
XDLRA--refs
2023-06-09 08:57 AM
XDLRA--repo-export
2023-06-09 08:57 AM
Citation
Components
Add Component
Link Projects
Add components to organize your project.
Tags
d9370ad0-3902-421a-8bf3-5d83d4611150
DataLad dataset
Add a tag
Recent Activity
Adina Svenja Wagner added file XDLRA--repo-export to OSF Storage in my-best-so-far
2023-06-09 08:57 AM
Adina Svenja Wagner added file XDLRA--refs to OSF Storage in my-best-so-far
2023-06-09 08:57 AM
Adina Svenja Wagner removed file XDLRA--refs from OSF Storage in my-best-so-far
2023-06-09 08:56 AM
Adina Svenja Wagner removed file XDLRA--repo-export from OSF Storage in my-best-so-far
2023-06-09 08:56 AM
Adina Svenja Wagner added file code/example.txt to OSF Storage in my-best-so-far
2023-06-09 08:56 AM
Adina Svenja Wagner created folder code in OSF Storage in my-best-so-far
2023-06-09 08:56 AM
< 1 2 3 >

Cloning the dataset

This dataset can now be cloned in the familiar way, using the OSF ID:

```
$ datalad clone osf://pmgx2/ my-best-yet
      install(ok): /tmp/my-best-yet
```

Its the best of both world!

1.4 Contributing

If you have any questions, comments, bug fixes or improvement suggestions, feel free to contact us via our Github page. Before contributing, be sure to read the contributing guidelines.

1.5 Acknowledgments

DataLad development is being performed as part of a US-German collaboration in computational neuroscience (CR-CNS) project "DataGit: converging catalogues, warehouses, and deployment logistics into a federated 'data distribution'" (Halchenko/Hanke), co-funded by the US National Science Foundation (NSF 1912266) and the German Federal Ministry of Education and Research (BMBF 01GQ1905). Additional support is provided by the German federal state of Saxony-Anhalt and the European Regional Development Fund (ERDF), Project: Center for Behavioral Brain Sciences, Imaging Platform.

DataLad is built atop the git-annex software that is being developed and maintained by Joey Hess.

The extension was created during the OHBM Hackathon 2020 and wouldn't have been possible without a dedicated team of volunteers.

1.6 Command line reference

`datalad-osf` has two main commands: `datalad osf-credentials` for OSF authentication management, and `datalad create-sibling-osf` for interactions with the OSF. Find out more about each command below.

1.6.1 datalad create-sibling-osf

Synopsis

```
datalad create-sibling-osf [-h] [--title TITLE] [-s NAME] [--storage-name NAME] [-d␣
↪DATASET] [--mode {annex|export|exportonly|gitonly}] [--existing MODE] [--trust-level␣
↪TRUST-LEVEL] [--tag TAG] [--public] [--category
↪{analysis|communication|data|hypothesis|instrumentation|methods and␣
↪measures|procedure|project|software|other}] [--description TEXT] [--version]
```

Description

Create a dataset representation at OSF.

This will create a node on OSF and initialize an osf special remote to point to it. There are two modes this can operate in: 'annex' and 'export'. The former uses the OSF node as a key-value store, that can be used by git-annex to copy data to and retrieve data from (potentially by any clone of the original dataset). The latter allows to use 'git annex export' to publish a snapshot of a particular version of the dataset. Such an OSF node will - in opposition to the 'annex' - be human-readable.

For authentication with OSF, you can define environment variables: Either 'OSF_TOKEN', or both 'OSF_USERNAME' and 'OSF_PASSWORD'. If neither of these is defined, the tool will fall back to the datalad credential manager and inquire for credentials interactively.

Options

-h, -\-help, -\-help-np

show this help message. –help-np forcefully disables the use of a pager for displaying the help message

-\-title *TITLE*

title of the to-be created OSF node that is displayed on the OSF website. Defaults to the basename of the root directory of the local dataset. Constraints: value must be a string or value must be NONE

-s *NAME*, -\-name *NAME*

Name of the to-be initialized osf-special-remote. Constraints: value must be a string [Default: 'osf']

-\-storage-name NAME

Name of the storage sibling (git-annex special remote). Must not be identical to the sibling name. If not specified, defaults to the sibling name plus '-storage' suffix. Constraints: value must be a string or value must be NONE

-d *DATASET*, -\-dataset *DATASET*

Dataset to create a sibling for. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-\-mode {annex|export|exportonly|gitonly}

[Default: 'annex']

-\-existing MODE

Action to perform, if a (storage) sibling is already configured under the given name and/or a target already exists. In this case, a dataset can be skipped ('skip'), or the command be instructed to fail ('error'). Constraints: value must be one of ('skip', 'error') or value must be NONE [Default: 'error']

-\-trust-level TRUST-LEVEL

specify a trust level for the storage sibling. If not specified, the default git-annex trust level is used. Constraints: value must be one of ('trust', 'semitrust', 'untrust') or value must be NONE

-\-tag TAG

specific one or more tags for the to-be-create OSF node. A tag 'DataLad dataset' and the dataset ID (if there is any) will be automatically added as additional tags. This option can be given more than once.

-\-public

make OSF node public.

-\-category {analysis|communication|data|hypothesis|instrumentation|methods and measures|procedure|project|software|other}

specific the OSF node category to be used for the node. The categorization determines what icon is displayed with the node on the OSF, and helps with search organization. Constraints: value must be one of ('analysis', 'communication', 'data', 'hypothesis', 'instrumentation', 'methods and measures', 'procedure', 'project', 'software', 'other') [Default: 'data']

-\-description TEXT

Description of the OSF node that will be displayed on the associated project page. By default a description will be generated based on the mode the sibling is put into. Constraints: value must be a string or value must be NONE

-\-version

show the module and its version which provides the command

Authors

datalad is developed by The DataLad Team and Contributors <team@datalad.org>.

1.6.2 datalad osf-credentials

Synopsis

```
datalad osf-credentials [-h] [--method {token|userpassword}] [--reset] [--version]
```

Description

Gather OSF credentials for subsequent non-interactive use

This command enables (re-)entry of OSF credentials for storage in a credential manager. Once credentials are known, they will be retrieved automatically on demand, and enable non-interactive use for the purpose of data transfer to and from OSF.

Credentials will be verified to enable successful authentication before being stored.

Options

-h, -\-help, -\-help-np

show this help message. –help-np forcefully disables the use of a pager for displaying the help message

-\-method {token|userpassword}

authentication method to use. 'token' authentication is strongly recommended. Constraints: value must be one of ('token', 'userpassword') [Default: 'token']

-\-reset

reset existing credentials and force re-entry.

-\-version

show the module and its version which provides the command

Authors

datalad is developed by The DataLad Team and Contributors <team@datalad.org>.

1.7 Python API

`datalad-osf` has two main commands that are exposed as functions via `datalad.api` and as methods of the `Dataset` class: `osf_credentials` for OSF authentication management, and `create_sibling_osf` for interactions with the OSF. Find out more about each command below.

osf_credentials([method, reset])	Gather OSF credentials for subsequent non-interactive use
create_sibling_osf([title, name, ...])	Create a dataset representation at OSF.

1.7.1 datalad.api.osf_credentials

datalad.api.**osf_credentials**(*method='token'*, *reset=False*)

Gather OSF credentials for subsequent non-interactive use

This command enables (re-)entry of OSF credentials for storage in a credential manager. Once credentials are known, they will be retrieved automatically on demand, and enable non-interactive use for the purpose of data transfer to and from OSF.

Credentials will be verified to enable successful authentication before being stored.

Parameters

- **method** (*{'token', 'userpassword'}, optional*) – authentication method to use. 'token' authentication is strongly recommended. [Default: 'token']
- **reset** (*bool, optional*) – reset existing credentials and force re-entry. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result

status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]

- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

1.7.2 datalad.api.create_sibling_osf

datalad.api.**create_sibling_osf**(*title=None*, *name='osf'*, *storage_name=None*, *dataset=None*, *mode='annex'*, *existing='error'*, *trust_level=None*, *tags=None*, *public=False*, *category='data'*, *description=None*)

Create a dataset representation at OSF.

This will create a node on OSF and initialize an osf special remote to point to it. There are two modes this can operate in: 'annex' and 'export'. The former uses the OSF node as a key-value store, that can be used by git-annex to copy data to and retrieve data from (potentially by any clone of the original dataset). The latter allows to use 'git annex export' to publish a snapshot of a particular version of the dataset. Such an OSF node will - in opposition to the 'annex' - be human-readable.

For authentication with OSF, you can define environment variables: Either 'OSF_TOKEN', or both 'OSF_USERNAME' and 'OSF_PASSWORD'. If neither of these is defined, the tool will fall back to the datalad credential manager and inquire for credentials interactively.

Parameters

- **title** (*str or None, optional*) – title of the to-be created OSF node that is displayed on the OSF website. Defaults to the basename of the root directory of the local dataset. [Default: None]
- **name** (*str, optional*) – Name of the to-be initialized osf-special-remote. [Default: 'osf']
- **storage_name** (*str or None, optional*) – Name of the storage sibling (git-annex special remote). Must not be identical to the sibling name. If not specified, defaults to the sibling name plus '-storage' suffix. [Default: None]
- **dataset** (*Dataset or None, optional*) – Dataset to create a sibling for. [Default: None]
- **mode** (*{'annex', 'export', 'exportonly', 'gitonly'}, optional*) – [Default: 'annex']
- **existing** (*{'skip', 'error'} or None, optional*) – Action to perform, if a (storage) sibling is already configured under the given name and/or a target already exists. In this case, a dataset can be skipped ('skip'), or the command be instructed to fail ('error'). [Default: 'error']
- **trust_level** (*{'trust', 'semitrust', 'untrust'} or None, optional*) – specify a trust level for the storage sibling. If not specified, the default git-annex trust level is used. [Default: None]
- **tags** – specific one or more tags for the to-be-create OSF node. A tag 'DataLad dataset' and the dataset ID (if there is any) will be automatically added as additional tags. . [Default: None]

- **public** (*bool, optional*) – make OSF node public. [Default: False]
- **category** (*{'analysis', 'communication', 'data', 'hypothesis', 'instrumentation', 'methods and measures', 'procedure', 'project', 'software', 'other'}, optional*) – specific the OSF node category to be used for the node. The categorization determines what icon is displayed with the node on the OSF, and helps with search organization. [Default: 'data']
- **description** (*str or None, optional*) – Description of the OSF node that will be displayed on the associated project page. By default a description will be generated based on the mode the sibling is put into. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

1.8 git-annex utilities

`datalad-osf` comes with a git-annex special remote implementation for the OSF. Find out more by clicking on the module name below.

annex_remote.OSFSpecialRemote(*args)	git-annex special remote for the open science framework

1.8.1 datalad_osf.annex_remote.OSFSpecialRemote

class datalad_osf.annex_remote.**OSFSpecialRemote**(**args*)

git-annex special remote for the open science framework

Any OSF node can be used as a remote, but the recommended setup is to create a subcomponent of your project for archiving your data. Mark it with the Data category so you can find it quickly and take note of its URL. Each component (or project) has a URL like https://osf.io/6zbyf/ which is needed to connect to it.

Todo: Write doc section on how to do that, and what URL should be used to configure the special remote.

Initialize the special remote:

```
git annex initremote osf type=external externaltype=osf \
     encryption=none node=<your-component-id>
```

To upload files you need to supply credentials.

Todo: Outline how this can be done

OSF_USERNAME, OSF_PASSWORD - credentials, OR OSF_TOKEN # TODO: untested, possibly currently broken in osfclient.

Because this is a special remote, the uploaded data do not retain the git folder structure.

The following parameters can be given to *git-annex initremote*, or *git annex enableremote* to (re-)configure a special remote.

node
the OSF URL of the file store

path
a subpath with (default: /)

See also:

https://git-annex.branchable.com/special_remotes
Documentation on git-annex special remotes

https://osf.io
Open Science Framework, a science-focused filesharing and archiving site.

__init__(**args*)

Methods

__init__(*args)	
checkpresent(key)	Report whether the OSF node has a particular key
checkpresentexport(key, remote_file)	Return if the file *remote_file* is present in the remote
checkurl(url)	Asks the remote to check if the url's content can currently be downloaded (without downloading it).
claimurl(url)	Asks the remote if it wishes to claim responsibility for downloading an url.
error(error_msg)	Generic error. Can be sent at any time if things get too messed up to continue. If the program receives an error() from git-annex, it can exit with its own error(). Eg.: self.annex.error("Error received. Exiting.") raise SystemExit.
exportsupported()	
getavailability()	Asks the remote if it is locally or globally available.
getcost()	Requests the remote to return a use cost.
initremote()	
listconfigs()	
prepare()	
remove(key)	Remove a key from the remote
removeexport(key, remote_file)	Remove the file in *remote_file* from the remote
removeexportdirectory(remote_directory)	Remove the directory *remote_directory* from the remote
renameexport(key, filename, new_filename)	Move the remote file in *name* to *new_name*
setup()	
transfer_retrieve(key, filename)	Get a key from OSF and store it to *filename*
transfer_store(key, filename)	
transferexport_retrieve(key, local_file, ...)	Get the file located at *remote_file* from the remote
transferexport_store(key, local_file, ...)	Store the file located at *local_file* to *remote_file* on the remote
whereis(key)	Asks the remote to provide additional information about ways to access the content of a key stored in it, such as eg, public urls.

Attributes

files

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

C

O