

---

# **datalad Documentation**

*Release 0.1*

**Datalad team**

**Jun 26, 2017**



---

## Contents

---

<b>1</b>	<b>Getting started</b>	<b>1</b>
<b>2</b>	<b>Change log</b>	<b>3</b>
<b>3</b>	<b>Acknowledgments</b>	<b>11</b>
<b>4</b>	<b>Concepts and technologies</b>	<b>13</b>
<b>5</b>	<b>Commands and API</b>	<b>31</b>
<b>6</b>	<b>Automated data distributions</b>	<b>173</b>
<b>7</b>	<b>Indices and tables</b>	<b>175</b>
	<b>Python Module Index</b>	<b>177</b>



### Installation

Datalad is a Python package and can be installed via [pip](#), which is the preferred method unless system packages are available for the target platform (see below):

```
pip install datalad
```

This will automatically install all software dependencies necessary to provide core functionality. Several additional installation schemes are supported (e.g., [publish](#), [metadata](#), [tests](#), [crawl](#)):

```
pip install datalad[SCHEME]
```

where `SCHEME` can be any supported scheme, such as the ones listed above.

In addition, it is necessary to have a working installation of [git-annex](#), which is not set up automatically at this point.

### (Neuro)Debian, Ubuntu, and similar systems

For Debian-based operating systems the most convenient installation method is to enable the [NeuroDebian](#) repository, and to install `datalad` as a system package:

```
sudo apt-get install datalad
```

### MacOSX

A simple way to get things installed is the [homebrew](#) package manager, which in itself is fairly easy to install. `Git-annex` is installed by the command:

```
brew install git-annex
```

Once Git-annex is available, datalad can be installed via `pip` as described above. `pip` comes with Python distributions like `anaconda`.

## First steps

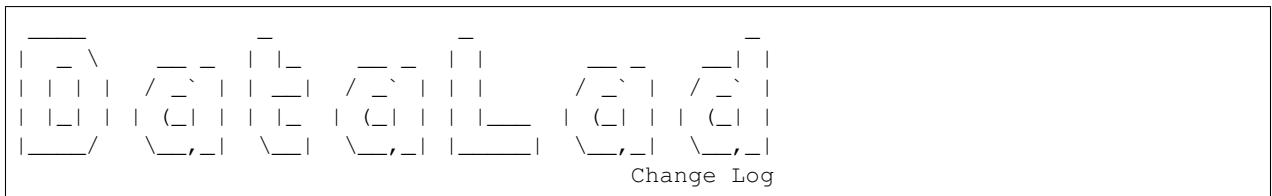
After datalad is installed it can be queried for information about known datasets. For example, we might want to look for dataset that were funded by, or acknowledge the US National Science Foundation (NSF):

```
~ % datalad search NSF
No DataLad dataset found at current location
Would you like to install the DataLad superdataset at '~/datalad'? (choices: yes, ↵
↵no): yes
2016-10-24 09:13:32,414 [INFO    ] Installing dataset at ~/datalad from http://
↵datasets.datalad.org/
From now on you can refer to this dataset using the label '///'
2016-10-24 09:13:39,072 [INFO    ] Performing search using DataLad superdataset '~/
↵datalad'
2016-10-24 09:13:39,086 [INFO    ] Loading and caching local meta-data... might take a ↵
↵few seconds
~/datalad/openfmri/ds000001
~/datalad/openfmri/ds000002
~/datalad/openfmri/ds000003
...
```

On first attempt, datalad offers assistance to obtain a *superdataset* with information on all datasets it knows about. This is a lightweight container that does not actually contain data, but meta information only. Once downloaded queries can be made offline.

Any known dataset can now be installed inside the local superdataset with a command like this:

```
datalad install ~/datalad/openfmri/ds000002
```



This is a very high level and scarce summary of the changes between releases. We would recommend to consult log of the [DataLad git repository](#) for more details ATM.

### 0.6.1 (???, 2017) – will be better than ever

I bet we will fix some bugs and make a world even a better place.

#### Major refactoring and deprecations

- `add-sibling` has been fully replaced by the `siblings` command
- `create-sibling`, and `unlock` have been rewriting to support the same common API as most other commands

#### Enhancements and new features

- `siblings` can now be used to query and configure a local repository by using the sibling name here
- `siblings` can now query and set annex preferred content configuration. This includes `wanted` (as previously supported in other commands), and now also `required`
- documentation for all commands is now built in a uniform fashion, leading to more...guess what...uniform documentation

- Significant parts of the documentation of been updated

### Fixes

- API documentation is now rendered properly as HTML, and is easier to browse by having more compact pages

### 0.6.0 (Jun 14, 2017) – German perfectionism

This release includes a **huge** refactoring to make code base and functionality more robust and flexible

- outputs from API commands could now be highly customized. See `--output-format`, `--report-status`, `--report-type`, and `--report-type` options for `datalad` command.
- effort was made to refactor code base so that underlying functions behave as generators where possible
- input paths/arguments analysis was redone for majority of the commands to provide unified behavior

### Major refactoring and deprecations

- `add-sibling` and `rewrite-urls` were refactored in favor of new `siblings` command which should be used for siblings manipulations
- `'datalad.api.alwaysrender'` config setting/support is removed in favor of new outputs processing

### Fixes

- Do not flush manually git index in pre-commit to avoid “Death by the Lock” issue
- Deployed by `publish post-update` hook script now should be more robust (tolerate directory names with spaces, etc.)
- A variety of fixes, see [list of pull requests and issues closed](#) for more information

### Enhancements and new features

- new `annotate-paths` plumbing command to inspect and annotate provided paths. Use `--modified` to summarize changes between different points in the history
- new `clone` plumbing command to provide a subset (install a single dataset from a URL) functionality of `install`
- new `[diff]` plumbing command
- new `siblings` command to list or manipulate siblings
- new `[subdatasets]` command to list subdatasets and their properties
- `drop` and `remove` commands were refactored
- `benchmarks/` collection of `Airspeed velocity` benchmarks initiated. See reports at <http://datalad.github.io/datalad/>
- crawler would try to download a new url multiple times increasing delay between attempts. Helps to resolve problems with extended crawls of Amazon S3
- `CRCNS` crawler pipeline now also fetches and aggregates meta-data for the datasets from datacite
- overall optimisations to benefit from the aforementioned refactoring and improve user-experience



- a few stub and not (yet) implemented commands (e.g. `move`) were removed from the interface
- Web frontend got proper coloring for the breadcrumbs and some additional caching to speed up interactions. See <http://datasets.datalad.org>
- Small improvements to the online documentation. See e.g. [summary of differences between git/git-annex/datalad](#)

### 0.5.1 (Mar 25, 2017) – cannot stop the progress

A bugfix release

#### Fixes

- `add` was forcing addition of files to annex regardless of settings in `.gitattributes`. Now that decision is left to annex by default
- `tools/testing/run_doc_examples` used to run doc examples as tests, fixed up to provide status per each example and not fail at once
- `doc/examples`
- `3rdparty_analysis_workflow.sh` was fixed up to reflect changes in the API of 0.5.0.
- progress bars
- should no longer crash **datalad** and report correct sizes and speeds
- should provide progress reports while using Python 3.x

#### Enhancements and new features

- `doc/examples`
- `nipype_workshop_dataset.sh` new example to demonstrate how new super- and sub- datasets were established as a part of our datasets collection

### 0.5.0 (Mar 20, 2017) – it's huge

This release includes an avalanche of bug fixes, enhancements, and additions which at large should stay consistent with previous behavior but provide better functioning. Lots of code was refactored to provide more consistent code-base, and some API breakage has happened. Further work is ongoing to standardize output and results reporting (see [PR 1350](#))

#### Most notable changes

- requires `git-annex`  $\geq$  6.20161210 (or better even  $\geq$  6.20161210 for improved functionality)
- commands should now operate on paths specified (if any), without causing side-effects on other dirty/staged files
- `save`
  - `-a` is deprecated in favor of `-u` or `--all-updates` so only changes known components get saved, and no new files automatically added

- `-S` does no longer store the originating dataset in its commit message
- `add`
  - can specify commit/save message with `-m`
- `add-sibling` and `create-sibling`
  - now take the name of the sibling (remote) as a `-s` (`--name`) option, not a positional argument
  - `--publish-depends` to setup publishing data and code to multiple repositories (e.g. `github + web-serve`) should now be functional see [this comment](#)
  - `got --publish-by-default` to specify what refs should be published by default
  - `got --annex-wanted`, `--annex-groupwanted` and `--annex-group` settings which would be used to instruct annex about preferred content. `publish` then will publish data using those settings if `wanted` is set.
  - `got --inherit` option to automagically figure out url/wanted and other git/annex settings for new remote sub-dataset to be constructed
- `publish`
  - `got --skip-failing` refactored into `--missing` option which could use new feature of `create-sibling --inherit`

## Fixes

- More consistent interaction through ssh - all ssh connections go through `sshrun` shim for a “single point of authentication”, etc.
- More robust `ls` operation outside of the datasets
- A number of fixes for direct and v6 mode of annex

## Enhancements and new features

- New `drop` and `remove` commands
- `clean`
  - `got --what` to specify explicitly what cleaning steps to perform and now could be invoked with `-r`
- `datalad` and `git-annex-remote*` scripts now do not use `setuptools` entry points mechanism and rely on simple `import` to shorten start up time
- `Dataset` is also now using `Flyweight pattern`, so the same instance is reused for the same dataset
- progressbars should not add more empty lines

## Internal refactoring

- Majority of the commands now go through `_prep` for arguments validation and pre-processing to avoid recursive invocations

### 0.4.1 (Nov 10, 2016) – CA release

Requires now `GitPython >= 2.1.0`

## Fixes

- `save`
  - to not save staged files if explicit paths were provided
- improved (but not yet complete) support for direct mode
- `update` to not crash if some sub-datasets are not installed
- do not log calls to `git config` to avoid leakage of possibly sensitive settings to the logs

## Enhancements and new features

- New `rfc822-compliant metadata` format
- `save`
  - `-S` to save the change also within all super-datasets
- `add` now has progress-bar reporting
- `create-sibling-github` to create a `:term:sibling` of a dataset on github
- `OpenfMRI` crawler and datasets were enriched with URLs to separate files where also available from `openfMRI s3` bucket (if upgrading your datalad datasets, you might need to run `git annex enableremote datalad` to make them available)
- various enhancements to log messages
- web interface
  - populates “install” box first thus making UX better over slower connections

### 0.4 (Oct 22, 2016) – Paris is waiting

Primarily it is a bugfix release but because of significant refactoring of the `install` and `get` implementation, it gets a new minor release.

## Fixes

- be able to `get` or `install` while providing paths while being outside of a dataset
- remote annex datasets get properly initialized
- robust detection of outdated `git-annex`

## Enhancements and new features

- interface changes
  - `get --recursion-limit=existing` to not recurse into not-installed subdatasets
  - `get -n` to possibly install sub-datasets without getting any data
  - `install --jobs | -J` to specify number of parallel jobs for annex `get` call could use (ATM would not work when data comes from archives)
- more (unit-)testing

- documentation: see <http://docs.datalad.org/en/latest/basics.html> for basic principles and useful shortcuts in referring to datasets
- various webface improvements: breadcrumb paths, instructions how to install dataset, show version from the tags, etc.

### 0.3.1 (Oct 1, 2016) – what a wonderful week

Primarily bugfixes but also a number of enhancements and core refactorings

#### Fixes

- do not build manpages and examples during installation to avoid problems with possibly previously outdated dependencies
- `install` can be called on already installed dataset (with `-r` or `-g`)

#### Enhancements and new features

- complete overhaul of datalad configuration settings handling (see [Configuration documentation](#)), so majority of the environment. Now uses git format and stores persistent configuration settings under `.datalad/config` and local within `.git/config` variables we have used were renamed to match configuration names
- `create-sibling` does not now by default upload web front-end
- `export` command with a plug-in interface and `tarball` plugin to export datasets
- in Python, `.api` functions with rendering of results in command line got a `_`-suffixed sibling, which would render results as well in Python as well (e.g., using `search_` instead of `search` would also render results, not only output them back as Python objects)
- `get`
  - `--jobs` option (passed to `annex get`) for parallel downloads
  - total and per-download (with `git-annex >= 6.20160923`) progress bars (note that if content is to be obtained from an archive, no progress will be reported yet)
- `install --reckless` mode option
- `search`
  - highlights locations and fieldmaps for better readability
  - supports `-d^` or `-d///` to point to top-most or centrally installed meta-datasets
  - “complete” paths to the datasets are reported now
  - `-s` option to specify which fields (only) to search
- various enhancements and small fixes to [meta-data](#) handling, `ls`, custom remotes, code-base formatting, downloaders, etc
- completely switched to `tqdm` library (`progressbar` is no longer used/supported)

### 0.3 (Sep 23, 2016) – winter is coming

Lots of everything, including but not limited to

- enhanced index viewer, as the one on <http://datasets.datalad.org>
- initial new data providers support: Kaggle, BALSAs, NDA, NITRC
- initial meta-data support and management
- new and/or improved crawler pipelines for BALSAs, CRCNS, OpenfMRI
- refactored `install` command, now with separate `get`
- some other commands renaming/refactoring (e.g., `create-sibling`)
- `datalad search` would give you an option to install `datalad`'s super-dataset under `~/datalad` if ran outside of a dataset

### 0.2.3 (Jun 28, 2016) – busy OHBM

New features and bugfix release

- support of `///` urls to point to <http://datasets.datalad.org>
- variety of fixes and enhancements throughout

### 0.2.2 (Jun 20, 2016) – OHBM we are coming!

New feature and bugfix release

- greatly improved documentation
- `publish` command API RFinng allows for custom options to `annex`, and uses `-to REMOTE` for consistent with `annex` invocation
- variety of fixes and enhancements throughout

### 0.2.1 (Jun 10, 2016)

- variety of fixes and enhancements throughout

### 0.2 (May 20, 2016)

Major RFinng to switch from relying on `rdflib` to `git` native submodules etc

### 0.1 (Oct 14, 2015)

Release primarily focusing on interface functionality including initial publishing



## CHAPTER 3

---

### 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 1429999) and the German Federal Ministry of Education and Research (BMBF 01GQ1411).

Datalad is built atop the [git-annex](#) software that is being developed and maintained by [Joey Hess](#).





## Background and motivation

### Vision

Data is at the core of science, and unobstructed access promotes scientific discovery through collaboration between data producers and consumers. The last years have seen dramatic improvements in availability of data resources for collaborative research, and new data providers are becoming available all the time.

However, despite the increased availability of data, their accessibility is far from being optimal. Potential consumers of these public datasets have to manually browse various disconnected warehouses with heterogeneous interfaces. Once obtained, data is disconnected from its origin and data versioning is often ad-hoc or completely absent. If data consumers can be reliably informed about data updates at all, review of changes is difficult, and re-deployment is tedious and error-prone. This leads to wasteful friction caused by outdated or faulty data.

The vision for this project is to transform the state of data-sharing and collaborative work by providing uniform access to available datasets – independent of hosting solutions or authentication schemes – with reliable versioning and versatile deployment logistics. This is achieved by means of a *dataset* handle, a lightweight representation of a dataset that is capable of tracking the identity and location of a dataset's content as well as carry meta-data. Together with associated software tools, scientists are able to obtain, use, extend, and share datasets (or parts thereof) in a way that is traceable back to the original data producer and is therefore capable of establishing a strong connection between data consumers and the evolution of a dataset by future extension or error correction.

Moreover, DataLad aims to provide all tools necessary to create and publish *data distributions* — an analog to software distributions or app-stores that provide logistics middleware for software deployment. Scientific communities can use these tools to gather, curate, and make publicly available specialized collections of datasets for specific research topics or data modalities. All of this is possible by leveraging existing data sharing platforms and institutional resources without the need for funding extra infrastructure of duplicate storage. Specifically, this project aims to provide a comprehensive, extensible data distribution for neuroscientific datasets that is kept up-to-date by an automated service.

## Technological foundation: git-annex

The outlined task is not unique to the problem of data-sharing in science. Logistical challenges such as delivering data, long-term storage and archiving, identity tracking, and synchronization between multiple sites are rather common. Consequently, solutions have been developed in other contexts that can be adapted to benefit scientific data-sharing.

The closest match is the software tool [git-annex](#). It combines the features of the distributed version control system (dVCS) [Git](#) — a technology that has revolutionized collaborative software development — with versatile data access and delivery logistics. Git-annex was originally developed to address use cases such as managing a collection of family pictures at home. With git-annex, any family member can obtain an individual copy of such a picture library — the *annex*. The annex in this example is essentially an image repository that presents individual pictures to users as files in a single directory structure, even though the actual image file contents may be distributed across multiple locations, including a home-server, cloud-storage, or even off-line media such as external hard-drives.

Git-annex provides functionality to obtain file contents upon request and can prompt users to make particular storage devices available when needed (e.g. a backup hard-drive kept in a fire-proof compartment). Git-annex can also remove files from a local copy of that image repository, for example to free up space on a laptop, while ensuring a configurable level of data redundancy across all known storage locations. Lastly, git-annex is able to synchronize the content of multiple distributed copies of this image repository, for example in order to incorporate images added with the git-annex on the laptop of another family member. It is important to note that git-annex is agnostic of the actual file types and is not limited to images.

We believe that the approach to data logistics taken by git-annex and the functionality it is currently providing are an ideal middleware for scientific data-sharing. Its data repository model *annex* readily provides the majority of principal features needed for a dataset handle such as history recording, identity tracking, and item-based resource locators. Consequently, instead of a from-scratch development, required features, such as dedicated support for existing data-sharing portals and dataset meta-information, can be added to a working solution that is already in production for several years. As a result, DataLad focuses on the expansion of git-annex's functionality and the development of tools that build atop Git and git-annex and enable the creation, management, use, and publication of dataset handles and collections thereof.

## Objective

Building atop git-annex, DataLad aims to provide a single, uniform interface to access data from various data-sharing initiatives and data providers, and functionality to create, deliver, update, and share datasets for individuals and portal maintainers. As a command-line tool, it provides an abstraction layer for the underlying Git-based middleware implementing the actual data logistics, and serves as a foundation for other future user front-ends, such as a web-interface.

## Delineation from related solutions

To our knowledge, there is no other effort with a scope as broad as DataLad's. Datalad aims to unify access to vast arrays of (scientific) data in a domain and data modality agnostic fashion with as few and universally available software dependencies as possible.

The most comparable project regarding the idea of federating access to various data providers is the [iRODS-based INCF Dataspace](#) project. IRODS is a powerful, NSF-supported framework, but it requires non-trivial deployment and management procedures. As a representative of *data grid* technology, it is more suitable for an institutional deployment, as data access, authentication, permission management, and versioning are complex and not-feasible to be performed directly by researchers. Datalad on the other hand federates institutionally hosted data, but in addition enables individual researchers and small labs to contribute datasets to the federation with minimal cost and without the need for centralized coordination and permission management.

## Data catalogs

Existing data-portals, such as [DataDryad](#), or domain-specific ones (e.g. [Human Connectome](#), [OpenfMRI](#)), concentrate on collecting, cataloging, and making data available. They offer an abstraction from local data management peculiarities (organization, updates, sharing). Ad-hoc collections of pointers to available data, such as [reddit datasets](#) and [Inside-R datasets](#), do not provide any unified interface to assemble and manage such data. Data portals can be used as seed information and data providers for DataLad. These portals could in turn adopt DataLad to expose readily usable data collections via a federated infrastructure.

## Data delivery/management middleware

Even though there are projects to manage data directly with dVCS (e.g. Git), such as the [Rdatasets Git repository](#) this approach does not scale, for example to the amount of data typically observed in a scientific context. Datalad uses [git-annex](#) to support managing large amounts of data with Git, while avoiding the scalability issues of putting data directly into Git repositories.

In scientific software development, frequently using Git for source code management, many projects are also confronted with the problem of managing large data arrays needed, for example, for software testing. An exemplar project is [ITK Data](#) which is conceptually similar to git-annex: data content is referenced by unique keys (checksums), which are made redundantly available through multiple remote key-store farms and can be obtained using specialized functionality in the CMake software build system. However, the scope of this project is limited to software QA, and only provides an ad-hoc collection of guidelines and supporting scripts.

The git-annex website provides a [comparison](#) of Git-annex to other available distributed data management tools, such as [git-media](#), [git-fat](#), and others. None of the alternative frameworks provides all of the features of git-annex, such as integration with native Git workflows, distributed redundant storage, and partial checkouts in one project. Additional features of git-annex which are not necessarily needed by DataLad (git-annex assistant, encryption support, etc.) make it even more appealing for extended coverage of numerous scenarios. Moreover, neither of the alternative solutions has already reached a maturity, availability, and level of adoption that would be comparable to that of git-annex.

## Git/Git-annex/DataLad

Although it is possible, and intended, to use DataLad without ever invoking git or git-annex commands directly, it is useful to appreciate that DataLad is build atop of very flexible and powerful tools. Knowing basics of git and git-annex in addition to DataLad helps to not only make better use of DataLad but also to enable more advanced and more efficient data management scenarios. Datalad makes use of lower-level configuration and data structures as much as possible. Consequently, it is possible to manipulate Datalad datasets with low-level tools if needed. Moreover, Datalad datasets are compatible with tools and services designed to work with plain Git repositories, such as the popular [GitHub](#) service.

To better illustrate the different scopes, the following table provides an overview of the features that are contributed by each software technology layer.

Feature	Git	Git-annex	DataLad
Version control (text, code)	✓	✓ can mix	✓ can mix
Version control (binary data)	(not advised)	✓	✓
Auto-crawling available resources		✓ RSS feeds	✓ flexible
Unified dataset handling			✓
<ul style="list-style-type: none"> <li>• recursive operation on datasets</li> </ul>			✓
<ul style="list-style-type: none"> <li>• seamless operation across datasets boundaries</li> </ul>			✓
<ul style="list-style-type: none"> <li>• meta-data support</li> </ul>		✓ per-file	✓
<ul style="list-style-type: none"> <li>• meta-data aggregation</li> </ul>			✓ flexible
Unified authentication interface			✓

## Basic principles

DataLad is designed to be used both as a command-line tool, and as a Python module. The sections [Command line reference](#) and [Python module reference](#) provide detailed description of the commands and functions of the two interfaces. This section presents common concepts. Although examples will frequently be presented using command line interface commands, all functionality with identically named functions and options are available through Python API as well.

## Datasets

A Datalad *dataset* is a Git repository that may or may not have a data *annex* that is used to manage data referenced in a dataset. In practice, most DataLad datasets will come with an annex.

Datasets can contain other datasets (*subdatasets*), which can in turn contain subdatasets, and so on. There is no limit to the depth of nesting datasets. Each dataset in such a hierarchy has its own annex and its own history. The parent or *superdataset* only tracks the specific state of a subdataset, and information on where it can be obtained. This is a powerful yet lightweight mechanism for combining multiple individual datasets for a specific purpose, such as the combination of source code repositories with other resources for a tailored application. In many cases DataLad can work with a hierarchy of datasets just as if it were a single dataset.

A superdataset can also be seen as a curated collection of datasets, for example, for a certain data modality, a field of science, a certain author, or some all from one project (maybe the resource for a movie production). The lightweight coupling between super and subdatasets enables scenarios where individual datasets are maintained by a disjoint set of people, and the dataset collection itself can be curated by a completely independent entity. Any individual dataset can be part of any number of such collections.

Benefitting from Git’s support for workflows based on decentralized “clones” of a repository, DataLad’s datasets can be (re-)published to a new location without losing the connection between the “original” and the new “copy”. This

is extremely useful for collaborative work, but also in more mundane scenarios such as data backup, or temporary deployment of a dataset on a compute cluster, or in the cloud. Using `git-annex`, data can also get synchronized across different locations of a dataset (*siblings* in DataLad terminology). Using metadata tags, it is even possible to configure different levels of desired data redundancy across the network of dataset, or to prevent publication of sensitive data to publicly accessible repositories. Individual datasets in a hierarchy of (sub)datasets need not be stored at the same location. Continuing with an earlier example, it is possible to post a curated collection of datasets, as a superdataset, on Github, while the actual datasets live on different servers all around the world.

## API principles

### URLs and shortcuts

In most places where DataLad accepts URLs as arguments these URLs can be regular `http` or `https` protocol URLs (e.g. `https://www.example.com/path`), but also SSH URLs, such as `ssh://me@localhost/path`. Additionally, DataLad supports SSH login style resource identifiers, such as `me@localhost:/path`. Besides these, the symbol `///` can be used to point to DataLad's canonical *superdataset* at `http://datasets.datalad.org/`, which provides an automated collection of datasets from various portals and sites (see *Automatic creation and maintenance of datasets by crawling external resources*). Here are some common examples in command line notation:

```
datalad install /// install canonical superdataset (alone, without subdatasets) in a datasets.datalad.org/ subdirectory under the current directory
```

```
datalad install -r ///openfmri install openfmri superdataset with a collection of all datasets available from http://openfmri.org as subdatasets in the openfmri/ subdirectory
```

```
datalad install -g -J3 -r ///labs/haxby install the superdataset of the collection of datasets released by the lab of Dr. James V. Haxby with all subdatasets, while fetching all data files using 3 parallel download processes.
```

### `-dataset` argument

All commands which operate with/on datasets (practically all commands) have a `dataset` argument (`-d` or `--dataset` in command line) which takes a path to the dataset that the command should operate on. If a dataset is identified this way then any relative path that is provided as an argument to the command will be interpreted as being relative to the topmost directory of that dataset. If no dataset argument is provided, relative paths are considered to be relative to the current directory.

There are also some useful pre-defined “shortcut” values for dataset arguments:

```
/// refers to the “canonical” dataset located under $HOME/datalad/. So running datalad install -d/// crcns will install the crcns subdataset under $HOME/datalad/crcns. This is the same as running datalad install $HOME/datalad/crcns.
```

```
^ topmost superdataset containing the dataset the current directory is part of. For example, if you are in $HOME/datalad/openfmri/ds000001/sub-01 and want to search metadata of the entire superdataset you are under (in this case ///), run datalad search -d^ [something to search].
```

### Commands `install` vs `get`

The `install` and `get` commands might seem confusingly similar at first. Both of them could be used to install any number of subdatasets, and fetch content of the data files. Differences lie primarily in their default behaviour and outputs, and thus intended use. Both `install` and `get` take local paths as their arguments, but their default behavior and output might differ;

- **install** primarily operates and reports at the level of **datasets**, and returns as a result dataset(s) which either were just installed, or were installed previously already under specified locations. So result should be the same if the same `install` command ran twice on the same datasets. It **does not fetch** data files by default
- **get** primarily operates at the level of **paths** (datasets, directories, and/or files). As a result it returns only what was installed (datasets) or fetched (files). So result of rerunning the same `get` command should report that nothing new was installed or fetched. It **fetches** data files by default.

In how both commands operate on provided paths, it could be said that `install == get -n`, and `install -g == get`. But `install` also has ability to install new datasets from remote locations given their URLs (e.g., `http://datasets.datalad.org/` for our super-dataset) and SSH targets (e.g., `[login@]host:path`) if they are provided as the argument to its call or explicitly as `--source` option. If `datalad install --source URL DESTINATION` (command line example) is used, then dataset from URL gets installed under PATH. In case of `datalad install URL` invocation, PATH is taken from the last name within URL similar to how `git clone` does it. If former specification allows to specify only a single URL and a PATH at a time, later one can take multiple remote locations from which datasets could be installed.

So, as a rule of thumb – if you want to install from external URL or fetch a sub-dataset without downloading data files stored under annex – use `install`. In Python API `install` is also to be used when you want to receive in output the corresponding Dataset object to operate on, and be able to use it even if you rerun the script. If you would like to fetch data (possibly while installing any necessary to be installed sub-dataset to get to the file) – use `get`.

## Data management use cases

### A typical data management workflow

In this demo we will look at how `datalad` can be used in a rather common data management workflow: A 3rd-party dataset is obtained to serve as input for an analysis. The data processing is then collaboratively performed by two colleagues. Upon completion the results are published alongside the original data for further consumption.

#### Build atop 3rd-party data

Now, meet Bob. Bob has just started in the lab and has never used the version control system `Git` before. The first thing he does, is to configure his identity as it will be used to track changes in the datasets he will be working with. This step only needs to be done once on his first day in the lab.

```
# enter Bob's home directory
HOME="$BOBS_HOME"
cd ~
git config --global --add user.name Bob
git config --global --add user.email bob@example.com
```

After this initial setup, Bob is ready to go and can create his first *dataset*.

```
datalad create myanalysis --description "my phd in a day"
cd myanalysis
```

A `datalad` dataset can contain other datasets. As any content of a dataset is tracked and its precise state is recorded, this is a powerful method to specify and later resolve data dependencies. In this example, Bob wants to work with structural MRI data from the [studyforrest project](#), a public brain imaging data resource. These data are made available through [GitHub](#), so Bob can simply install the relevant dataset from this service and into his own dataset:

```
datalad install -d . --source https://github.com/psychoinformatics-de/studyforrest-
↳data-structural.git src/forrest_structural
```

and see that the `forrest_structural` was registered as a git submodule, which is a *subdataset* of his `myanalysis` dataset, but no data was fetched (`datalad ls -L` provides `size_installed/total_size` column):

```
# mostly for a test
grep src/forrest_structural .gitmodules
# to demonstrate ls
datalad ls -r -L .
```

Bob has decided to collect all data inputs for his project in a subdirectory `src/`, to make it obvious which parts of his analysis steps and code require 3rd-party data. Upon completion of the above command, Bob has now access to the entire dataset content, and precise current version of that dataset got linked to his `myanalysis`. However, no data was actually downloaded (yet). Datalad datasets primarily contain information on a dataset's content and where to obtain it, hence the installation above was done rather quickly, and will still be relatively lean even for a dataset that contains several hundred GBs of data.

For his first steps Bob just needs a single file of the dataset. In order to make it available locally, Bob can use the `get` command, and `datalad` will obtain requested data files from a remote data provider.

```
datalad get src/forrest_structural/sub-01/anat/sub-01_T1w.nii.gz
# just test data for now, could be
#datalad get src/forrest_structural/sub-*/anat/sub-*_T1w.nii.gz
```

Although we originally installed the dataset from Github, the actual data is hosted elsewhere. Datalad supports multiple redundant data providers per each file in a dataset, and will transparently attempt to obtain data from an alternative location if a particular data provider is not available.

Bob wants his analysis to be easily reproducible, and therefore manages his analysis scripts in the same dataset repository as the input data. Managing input data, analysis code, and results the same version control system creates a precise record of what version of code and input data was used to create which particular results. Datalad datasets are regular `Git` repositories and therefore provide the same powerful source code management features, as any other `Git` repository, and make them available for data too.

Bob decided to adopt the convention to collect all of his analysis code in a subdirectory `code/` in the root of his dataset. His first “analysis” script is rather simple:

```
mkdir code
echo "nib-ls src/forrest_structural/sub-01/anat/sub-*_T1w.nii.gz > result.txt" > code/
↳run_analysis.sh
```

In order to definitively document which data file his analysis needs at this point, Bob creates a second script that can (re-)obtain the required files:

```
echo "datalad get src/forrest_structural/sub-01/anat/sub-01_T1w.nii.gz" > code/get_
↳required_data.sh
```

In the future, this won't be necessary anymore as `datalad` itself will be able to record this information upon request.

At this point Bob is satisfied with his initial progress. He wants to record this precise state. In order to do that, Bob needs to make his just created scripts a part of his dataset. Again the `install` command is used for this purpose. However, Bob doesn't just want `datalad` to track these files and facilitate future downloads. He wants all `Git` features for working with them, so he adds them directly to the `Git` repository underlying his dataset.

```
# add all content in the code/ directory directly to git
datalad add --to-git code
```

At this point, `datalad` is aware of all changes that were made to the dataset and all the changes Bob made were automatically recorded, as you could easily check with `git log` command.

As Bob's analysis is completely scripted, he can now run it in full:

```
bash code/get_required_data.sh
bash code/run_analysis.sh
```

and add generated results to the dataset but to not save automatically this addition, so he could provide a custom message to depict a better description of the accomplished work:

```
datalad add --nosave result.txt

# and save current state
datalad save -m "First analysis results"
# git log
```

### Local collaboration

Some time later, Bob needs help with his analysis. He turns to his colleague Alice for help. Alice and Bob both work on the same computing server. Alice initially went through a similar configuration procedure of her Git identity as Bob.

```
HOME="$ALICES_HOME"
cd
git config --global --add user.name Alice
git config --global --add user.email alice@example.com
```

Bob has told Alice in which directory he keeps his analysis dataset. The colleagues' directories are configured to have permissions that allow for read-access for all lab-member, so Alice can obtain Bob's work directly from his home directory, including the `studyforrest-structural` *subdataset* he had:

```
# TODO: needs to get --description to avoid confusion
datalad install -r --source "$BOBS_HOME/myanalysis" bobs_analysis
cd bobs_analysis
```

At this point, Alice has a complete copy of Bob's entire dataset in the exact same state that Bob last saved. She is free to make any changes without affecting Bob's version of the dataset. Initially, all the datasets are as lightweight as possible.

With the script Bob created, Alice can obtain all required data content. Datalad knows that necessary file is available in Bob's version of the dataset on the same machine, so it won't even attempt to download it from its original location.

```
bash code/get_required_data.sh
```

Likewise, Alice can use datalad to obtain the results that Bob had generated.

```
datalad get result.txt
#cat result.txt
```

She can modify Bob's code to help him with his analysis...

```
echo "nib-ls src/forrest_structural/sub-*/anat/sub-*_T1w.nii.gz > result.txt" > code/
↪run_analysis.sh
```

... and execute it.



```
# `|| true` is only there for the purpose of testing this script
bash code/run_analysis.sh || true
```

However, when she performs actions that attempt to modify data files managed by datalad she will get an error. Datalad, by default, prevents modification of data files. If modification is desired (as in this case), datalad can *unlock* individual files, or the entire dataset. Afterwards modifications are possible.

```
# unlock the entire dataset
datalad unlock
bash code/run_analysis.sh
```

Once Alice is satisfied with her modifications she can save the new state.

```
# -a make datalad auto-detect modifications
datalad save -u -m "Alice always helps"
```

## Full circle

Now that Alice has improved Bob's analysis, Bob wants to obtain the changes she made. To achieve that, he registers Alice's version of the dataset as a *sibling*. As both are working on the same machine, Bob can just point to the respective directory, but it would also be possible to refer to a dataset via an http URL, or an SSH login and path.

```
HOME="$BOBS_HOME"
cd ~/myanalysis
datalad siblings add -s alice --url "$ALICES_HOME/bobs_analysis"
```

Once registered, Bob can update his dataset based on Alice's version, and merge here changes with his own.

```
datalad update -s alice --merge
```

He can, once again, use the `get` command to obtain the latest version of data files to get access to data contributed by Alice.

```
datalad get result.txt
```

## Going public

Lastly, let's assume that Bob completed his analysis and he is ready to share the results with the world, or a remote collaborator. One way to make datasets available, is to upload them to a webserver via SSH. Datalad supports this by creating a *sibling* for the dataset on the server, to which the dataset can be published (repeatedly).

```
# this generated sibling for the dataset and all subdatasets
datalad create-sibling --recursive -s public "$SERVER_URL"
```

Once the remote sibling is created and registered under the name "public", Bob can publish his version to it.

```
datalad publish -r --to public .
```

This command can be repeated as often as desired. Datalad checks the state of both the local and the remote sibling and transmits the changes.

## Creating a “new” derived dataset for Nipype workshop

For more information about the workshop, please visit <http://nipype.org/workshops/2017-03-boston/index.html> . As we will present git-annex and DataLad, I have decided to prepare a DataLad *dataset* from the tarball Satrajit Ghosh has shared URL to. That tarball is a trimmed down version of OpenfMRI ds000114 dataset which we also crawl and provide as a DataLad dataset.

### Deriving (cloning) a dataset

I have decided to first create a *superdataset* for the workshop (may be more datasets besides ds114 will be later) outside of our master *superdataset* which we distribute from <http://datasets.datalad.org> . So I just created a new *dataset* in a random directory. I wanted our ds114 dataset to be “derived” from original openfmri ds000114 dataset so we could readily reuse all the knowledge git-annex has about where files might be coming from. To achieve that I have just installed existing ds000114 dataset into our *superdataset*:

```
# Create dataset
datalad create --no-annex nipype-workshop-2017
cd nipype-workshop-2017
datalad install -d . ///openfmri/ds000114
cd ds000114
```

To make both original and this derived dataset accessible from the same repo I generated a detached branch (since I did not know at that point on which version of openfmri it is based on). And then added content from the tarball available from the OSF.

A few tricky points which you do not necessarily would run into in a more typical workflow:

- since branch is detached, it would be empty to start with, but we want to preserve the settings within `.gitattributes` (such as git annex backend). I could have just `git add .gitattributes` after `checkout --orphan` but I haven’t thought about that and created a new file from scratch.
- Original openfmri dataset did not have settings within `.gitattributes` to add all text files straight into git, so I have added those settings within a new `.gitattributes` . For more about settings git-annex understands as to what files should it handled (*largefiles*) or otherwise just pass to git to handle see <https://git-annex.branchable.com/tips/largefiles/> .

```
cat .gitattributes # sneak preview
git checkout --orphan nipype_test1 # generate a new detached branch
git clean -dfx # remove
git reset --hard # everything, to end up with super clean directory

# I did vim .gitattributes, but replacing here with echo
echo -e "* annex.backend=MD5E
* annex.largefiles=(not (mimetype=text/*))
" > .gitattributes

datalad add --to-git .gitattributes # storing this file within git, default commit_
↪msg
```

### Adding new content from a tarball off the web

Next goal was to download and add to annex the tarball Satra prepared for the workshop, and add its content under git-annex control. In datalad we have ‘download-url’ command, BUT unfortunately it has failed to download via https for this website (see [issue 1416](#) if resolved already) So I have reverted to using git annex directly which uses wget which worked out correctly

```
# download (~800MB) and add that file under git-annex without
git annex addurl --file=ds114_test1_with_freesurfer.tar.gz "$URL"

datalad save -m "Downloaded the tarball with derived data into annex"
datalad add-archive-content --delete --strip-leading-dirs ds114_test1_with_freesurfer.
↪tar.gz
```

Above `add-archive-content` command extracted content from the archive, stripping leading directory, and added all extracted files under `git/git-annex` using those rules specified in `.gitattributes` file:

- use MD5E (annex keys are based on md5 checksum with extension appended) backend
- add text files directly under git control, so only binary files are added under annex control and the entire repository's `.git/objects` is only around 30MB while pointing to all openfMRI releases, and this derived data

## Peering inside

Because I have reused original `///openfMRI/ds000114` dataset, I have gained knowledge about all the files which originated from that dataset. E.g. compare output of `whereis` command on `sub-01/anat` (which is also available from original openfMRI) and derivatives:

```
# in case of a fake tarball, output will not be very interesting
git annex whereis sub-01/anat
git annex whereis derivatives
```

and you can see that derivatives are available only locally or from “magical” `datalad-archives` remote which refers to the original tarball. So, even if we drop those files locally, they could get extracted from the tarball. And even if you do not have a tarball, `git-annex` would happily first download it from the OSF website for you.

## Adding dataset into bigger dataset

Having succeeded with construction of the dataset, I have decided to share it as a part of our bigger super dataset at <http://datasets.datalad.org>. This dataset was the first workshop dataset which was not part of some bigger collection, so I have decided to establish a new `subdataset workshops` within it, and move our `nipy` workshop superdataset into it.

```
cd $SUPERDATASET
# since in demo we do not have anything there, let's clone our superdataset
datalad install ///
cd datasets.datalad.org

# -redone because now datasets.datalad.org already has workshops dataset
# and datalad should refuse to create a new one (without removing old one first)
datalad create -d . workshops-redone # create subdataset to hold various workshops_
↪datasets
cd workshops-redone
mv "$TOPDIR/nipy-workshop-2017" nipy-2017 # chose shorter name
# add it as a subdataset (git submodule) within
datalad add -d . nipy-2017
```

## Adding meta-data descriptors for the dataset(s)

If you ever ran `datalad search` you know that one of the goals of DataLad is to use *Meta data* associated with the datasets.

```
# created some dataset_description.json (following BIDS format lazy me)
echo '{"Name": "Datasets for various workshops"}' > dataset_description.json
# add that file to git
datalad add --to-git --nosave dataset_description.json
# discover and aggregate all meta-data within workshops
datalad aggregate-metadata --guess-native-type --nosave -r
# and finally save all accumulated changes from above commands
# while also updating the topmost superdataset about this changes under 'workshops'
datalad save -S -m "Added dataset description and aggregated meta-data" -r
# go upstairs and aggregate meta-information across its direct datasets without_
↳recursing
# (since might take awhile)
cd ..
datalad aggregate-metadata --guess-native-type
```

### Publishing

NB instructions here might diverge a little from what was actually performed

Now it was time to publish this dataset as a part of our larger super-dataset. Because our demo superdataset is just a clone (or *sibling*) of original one, it does not have information about where it must be published to. So we first can create a sibling on remote server where we want also to deploy our web-frontend and then create similar siblings for every

```
datalad create-sibling --shared all \
  -s public --ui=true \
  --publish-by-default 'refs/heads/*' \
  --publish-by-default 'refs/tags/*' \
  "$PUBLISHLOC"

datalad create-sibling -s public --inherit -r --existing skip
```

By default DataLad does not publish any data, and in above create-sibling we also did not provide any *annex-wanted* settings to instruct annex about what data should be published to our public sibling. So I decided to provide additional instructions for annex directly about what data files I want to be published online from our website. Since original files under *sub-\** subdirectories are available from original OpenfMRI S3 bucket, we really needed to publish only *derivatives/\** files, which we can describe via

```
git -C workshops-redone/nipype-2017/ds000114 annex wanted public 'include=derivatives/
↳*'
```

And now I was ready to publish changes to the entire collection of datasets with a set of files we decided to share

```
datalad publish -r --to=public
```

Above commands created empty repositories for all the datasets we have locally and now I was ready to “publish” our datasets... Just a few final touches

There is a <”shortcoming” <https://github.com/datalad/datalad/issues/1428>>\_\_ which was discovered just now, because it was the first time we published datasets from non-master branch (*nipype\_test1*). Default branch on the remote where we published is master, so we need to checkout *nipype\_test1* branch and re-run out hooks/post-update hook to re-generate meta-data for dataset listing on web-frontend. Hopefully this portion of explanation will disappear with DataLad 0.5.1 or later ;-)

```
(
  cd $PUBLISHDIR/workshops-redone/nipype-2017/ds000114
```

```

git checkout nipype_test1
# rerun the hook to regenerate meta-data for web-frontend
cd .git; hooks/post-update
)

```

If I do any future changes, and save them, it should be sufficient to just rerun this *publish* command (possibly even without explicit `-to=public`) and have all datasets updated online, with data files under *derivatives/* in that repository posted as well.

## Browsing

If the location where we published our datasets is served by any http server, they now could be used from that location by others, while having complete history of changes stored in annex, and data files available either from that location or from original openfMRI S3 bucket.

If you do not have published to location served by a web server, as the case in our demo script, we could easily start one using the one which comes with Python:

```

cd "$PUBLISHDIR"
# Starting webserver
python -m SimpleHTTPServer 8080 1>/dev/null 2>&1 &
# we started webserver and can browse
PUBLISHURL=http://localhost:8080
browser=$(which x-www-browser 2>/dev/null)
if $browser; then
    echo "Opening browser to visit $PUBLISHURL which would allow to browse
↪$PUBLISHDIR content"
    $browser $PUBLISHURL &
else
    echo "Visit http://localhost:8080 in your browser."
fi
echo "

On that page
Press Enter when you want to finish
"

in=$(read)
kill %2 && echo "stopped browser(?)" || : # killing our browser job if any
kill %1 && echo "stopped server"

```

Since DataLad datasets are just git/git-annex repositories, we could as well publish them to multiple locations, including github.com, only without data. See `datalad-create-sibling-github` and `-publish-depends` option to instruct to publish first to our public http server which will host the data and then to github.com for more visibility and collaboration.

## Meta data

### Overview

Datalad has built-in, modular, and extensible support for meta data in various formats. The core concept is that meta data is accessed via dedicated parsers in their native format, avoiding the need for mandatory conversion into a “standard” format. Via these parser datalad is capable of performing a certain amount of meta data homogenization, and standardization into a JSON-LD compliant [linked data](#) structure for the purpose of meta data aggregation in

*superdatasets*. Through this mechanism it is possible to obtain and query meta data of any number of *subdatasets* without the need to actually install them.

### Sample datasets with meta data

<http://datasets.datalad.org> superdataset contains a collection of datasets which we have prepared primarily from available online data resources such as *OpenfMRI*, *CRCNS*, etc. Many of those datasets came with meta data in their native formats, such as *Brain Imaging Data Structure (BIDS)*. DataLad has *aggregated* metadata where it was available to enable basic *search* queries. If you run *search* command outside of any datalad dataset, it will offer to install our <http://datasets.datalad.org> superdataset at `~/datalad` and then search through its metadata. If that superdataset is already installed (by *datalad search* or manually via *datalad install -s /// ~/datalad*), you can refer to it in the search command using `-d ///` option, e.g.:

```
$> datalad search -d /// bids
/home/yoh/datalad/openfmri/ds000017A
/home/yoh/datalad/openfmri/ds000017
/home/yoh/datalad/dicoms/dartmouth-phantoms/bids_test3
/home/yoh/datalad/labs
/home/yoh/datalad/labs/haxby
/home/yoh/datalad/labs/haxby/raiders
/home/yoh/datalad/openfmri
/home/yoh/datalad/openfmri/ds000001
.. .
```

### Supported meta data formats

This following sections provide an overview of supported meta data formats.

#### RFC822-compliant meta data

This is a custom meta data format, inspired by the standard used for Debian software packages that is particularly suited for manual entry. This format is a good choice when meta data describing a dataset as a whole cannot be obtained from some other structured format. The syntax is **RFC 822**-compliant. In other words: this is a text-based format that uses the syntax of email headers. Meta data must be placed in `DATASETROOT/.datalad/meta.rfc822` for this format.

Here is an example:

```
Name: myamazingdataset
Version: 1.0.0-rc3
Description: Basic summary
  A text with arbitrary length and content that can span multiple
  .
  paragraphs (this is a new one)
License: CC0
  The person who associated a work with this deed has dedicated the work to the
  public domain by waiving all of his or her rights to the work worldwide under
  copyright law, including all related and neighboring rights, to the extent
  allowed by law.
  .
  You can copy, modify, distribute and perform the work, even for commercial
  purposes, all without asking permission.
Homepage: http://example.com
Funding: Grandma's and Grandpa's support
```

```
Issue-Tracker: https://github.com/datalad/datalad/issues
Cite-As: Mike Author (2016). We made it. The breakthrough journal of unlikely
events. 1, 23-453.
DOI: 10.0000/nothere.48421
```

The following fields are supported:

**Audience:** A description of the target audience of the dataset.

**Author:** A comma-delimited list of authors of the dataset, preferably in the format. Firstname Lastname  
<Email Adress>

**Cite-as:** Instructions on how to cite the dataset, or a structured citation.

**Description:** Description of the dataset as a whole. The first line should represent a compact short description with no more than 6-8 words.

**DOI:** A [digital object identifier](#) for the dataset.

**Funding:** Information on potential funding for the creation of the dataset and/or its content. This field can also be used to acknowledge non-monetary support.

**Homepage:** A URL to a project website for the dataset.

**Issue-tracker:** A URL to an issue tracker where known problems are documented and/or new reports can be submitted.

**License:** A description of the license or terms of use for the dataset. The first lines should contain a list of license labels (e.g. CC0, PDDL) for standard licenses, if possible. Full license texts or term descriptions can be included.

**Maintainer:** Can be used in addition and analog to `Author`, when authors (creators of the data) need to be distinguished from maintainers of the dataset.

**Name:** A short name for the dataset. It may be beneficial to avoid special characters, umlauts, spaces, etc. to enable widespread use of this name for URL, catalog keys, etc. in unmodified form.

**Version:** A version for the dataset. This should be in a format that is alphanumerically sortable and lead to a “greater” version for an update of a dataset.

## Brain Imaging Data Structure (BIDS)

Datalad has basic support for extraction of meta data from the [BIDS dataset\\_description.json](#) file.

## Friction-less data packages

Datalad has basic support for extraction of meta data from [friction-less data packages \(datapackage.json\)](#) file.

## JSON-LD meta data format

Datalad uses [JSON-LD](#) as its primary meta data format. By default, the following context (available from here) is used for any meta data item:

```
{
  "@context": {
    "@vocab": "http://schema.org/",
    "doap": "http://usefulinc.com/ns/doap#",
    "Author": {"@id": "schema:author"},
    "Audience": {"@id": "doap:audience"},
```

```

"Citation": {"@id": "schema:citation"},
"Contributors": {"@id": "schema:contributor"},
"Description": {"@id": "schema:description"},
"Homepage": {"@id": "doap:homepage"},
"IssueTracker": {"@id": "doap:bug-database"},
"Keywords": {"@id": "schema:keywords"},
"License": {"@id": "http://www.w3.org/1999/xhtml/vocab#license"},
"Location": {"@id": "schema:location"},
"Maintainer": {"@id": "doap:maintainer"},
"Name": {"@id": "schema:name"},
"ShortDescription": {"@id": "doap:shortdesc"},
"Type": {"@id": "schema:type"},
"Version": {"@id": "doap:Version"},
"conformsTo": {"@id": "dcterms:conformsTo"},
"fundedBy": {"@id": "foaf:fundedBy"},
"hasPart": {"@id": "dcterms:hasPart"},
"isPartOf": {"@id": "dcterms:isPartOf"},
"isVersionOf": {"@id": "dcterms:isVersionOf"},
"modified": {"@id": "dcterms:modified"},
"sameAs": {"@id": "schema:sameAs"}
}
}

```

While it is technically possible to mix different contexts across items this has not been fully tested yet. The following sections describe details and changes in the meta data specifications implemented in datalad.

### v0.1

- Original implementation

## Frequently asked questions

**I have no permission to install software on a server I want to use datalad on. Can I make it work nevertheless?**  
 pip supports installation into a user's home directory with `--user`. Git-annex, on the other hand, can be deployed by extracting pre-built binaries from a tarball (that also includes an up-to-date Git installation). [Obtain the tarball](#), extract it, and set the PATH environment variable to include the root of the extracted tarball. Fingers crossed and good luck!

## Glossary

Datalad purposefully uses a terminology that is different from the one used by its technological foundations [Git](#) and [git-annex](#). This glossary provides definitions for terms used in the datalad documentation and API, and relates them to the corresponding [Git/git-annex](#) concepts.

**annex** Extension to a [Git](#) repository, provided and managed by [git-annex](#) as means to track and distribute large (and small) files without having to inject them directly into a [Git](#) repository (which would slow [Git](#) operations significantly and impair handling of such repositories in general).

**dataset** A regular [Git](#) repository with an (optional) *annex*.

**sibling** A *dataset* (location) that is related to a particular dataset, by sharing content and history. In [Git](#) terminology, this is a *clone* of a dataset that is configured as a *remote*.



**subdataset** A *dataset* that is part of another dataset, by means of being tracked as a [Git](#) submodule. As such, a subdataset is also a complete dataset and not different from a standalone dataset.

**superdataset** A *dataset* that contains at least one *subdataset*.



## Command line reference

### Main command

#### **datalad**

#### Synopsis

```
datalad [-h] [-l LEVEL] [--pbs-runner {condor}] [-C PATH] [--version] [--dbg] [--
↳idbg] [-c KEY=VALUE] [--output-format {default,json,json_pp,tailored,'<template>'}]
↳[--report-status {success,failure,ok,notneeded,impossible,error}] [--report-type
↳{dataset,file}] [--on-failure {ignore,continue,stop}] {create,install,get,add,
↳publish,uninstall,drop,remove,update,create-sibling,create-sibling-github,unlock,
↳save,export,search,metadata,aggregate-metadata,test,crawl,crawl-init,ls,clean,add-
↳archive-content,download-url,annotate-paths,clone,create-test-dataset,diff,siblings,
↳sshrun,subdatasets} ...
```

#### Description

DataLad provides a unified data distribution with the convenience of git-annex repositories as a backend. DataLad command line tools allow to manipulate (obtain, create, update, publish, etc.) datasets and their collections.

#### *Commands for dataset operations*

- create** Create a new dataset from scratch
- install** Install a dataset from a (remote) source
- get** Get any dataset content (files/directories/subdatasets)
- add** Add files/directories to an existing dataset

**publish** Publish a dataset to a known sibling  
**uninstall** Uninstall subdatasets  
**drop** Drop file content from datasets  
**remove** Remove components from datasets  
**update** Update a dataset from a sibling  
**create-sibling** Create a dataset sibling on a UNIX-like SSH-accessible machine  
**create-sibling-github** Create dataset sibling on Github  
**unlock** Unlock file(s) of a dataset  
**save** Save the current state of a dataset  
**export** Export a dataset to another representation

*Commands for meta data handling*

**search** Search within available in datasets' meta data  
**metadata** Metadata manipulation for files and whole datasets  
**aggregate-metadata** Aggregate meta data of a dataset for later query

*Miscellaneous commands*

**test** Run internal DataLad (unit)tests  
**crawl** Crawl online resource to create or update a dataset  
**crawl-init** Initialize crawling configuration  
**ls** List summary information about URLs and dataset(s)  
**clean** Clean up after DataLad (possible temporary files etc.)  
**add-archive-content** Add content of an archive under git annex control  
**download-url** Download content

*Plumbing commands*

**annotate-paths** Analyze and act upon input paths  
**clone** Obtain a dataset copy from a URL or local source (path)  
**create-test-dataset** Create test (meta-)dataset  
**diff** Report changes of dataset component between revisions  
**siblings** Manage sibling configuration  
**sshrun** Run command on remote machines via SSH  
**subdatasets** Report subdatasets and their properties

*General information*

Detailed usage information for individual commands is available via command-specific `-help`, i.e.: `datalad <command> -help`

## Options

{create,install,get,add,publish,uninstall,drop,remove,update,create-sibling,create-sibling-github,unlock,save,export,search,metadata,aggregate-metadata,test,crawl,crawl-init,ls,clean,add-archive-content,download-url,annotate-paths,clone,create-test-dataset,diff,siblings,sshrun,subdatasets}

### **-h, --help, --help-np**

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

### **-l LEVEL, --log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-C PATH**

run as if datalad was started in <path> instead of the current working directory. When multiple `-C` options are given, each subsequent non-absolute `-C <path>` is interpreted relative to the preceding `-C <path>`. This option affects the interpretations of the path names in that they are made relative to the working directory caused by the `-C` option

### **--version**

show the program's version and license information

### **--dbg**

enter Python debugger when uncaught exception happens

### **--idbg**

enter IPython debugger when uncaught exception happens

### **-c KEY=VALUE**

configuration variable setting. Overrides any configuration read from a file, but is potentially overridden itself by configuration variables in the process environment.

**–output-format {default,json,json\_pp,tailored,<template>}**

select format for returned command results. ‘default’ give one line per result reporting action, status, path and an optional message; ‘json’ renders a JSON object with all properties for each result (one per line); ‘json\_pp’ pretty-prints JSON spanning multiple lines; ‘tailored’ enables a command-specific rendering style that is typically tailored to human consumption (no result output otherwise), ‘<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).

**–report-status {success,failure,ok,notneeded,impossible,error}**

constrain command result report to records matching the given status. ‘success’ is a synonym for ‘ok’ OR ‘notneeded’, ‘failure’ stands for ‘impossible’ OR ‘error’.

**–report-type {dataset,file}**

constrain command result report to records matching the given type. Can be given more than once to match multiple types.

**–on-failure {ignore,continue,stop}**

when an operation fails: ‘ignore’ and continue with remaining operations, the error is logged but does not lead to a non-zero exit code of the command; ‘continue’ works like ‘ignore’, but an error causes a non-zero exit code; ‘stop’ halts on first failure and yields non-zero exit code. A failure is any result with status ‘impossible’ or ‘error’.

“Control Your Data”

## Authors

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

## Dataset operations

### datalad-add

#### Synopsis

```
datalad-add [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d PATH] [--to-git] ↵  
↪ [--nosave] [-m MESSAGE] [-r] [--recursion-limit LEVELS] [-S] [--git-opts STRING] [--  
↪ annex-opts STRING] [--annex-add-opts STRING] [-J NJOBS] PATH [PATH ...]
```

#### Description

Add files/directories to an existing dataset.

Typically, files and directories to be added to a dataset would be placed into a directory of a dataset, and subsequently this command can be used to register this new content with the dataset. With recursion enabled, files will be added to their respective subdatasets as well.

By default all files are added to the dataset's annex, i.e. only their content identity and availability information is tracked with Git. This results in lightweight datasets. If desired, the `--to-git` flag can be used to tell datalad to inject files directly into Git. While this is not recommended for binary data or large files, it can be used for source code and meta-data to be able to benefit from Git's track and merge capabilities. Files checked directly into Git are always and unconditionally available immediately after installation of a dataset.

**NOTE** Power-user info: This command uses `git annex add`, or `git add` to incorporate new dataset content.

## Options

### PATH

path/name of the component to be added. The component must either exist on the filesystem already, or a SOURCE has to be provided. Constraints: value must be a string [Default: None]

### `--version`

show the program's version and license information

### `-h, --help, --help-np`

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

### `-l LEVEL, --log-level LEVEL`

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### `--pbs-runner {condor}`

execute command by scheduling it via available PBS. For settings, config file will be consulted

### `-d PATH, --dataset PATH`

specify the dataset to perform the add operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### `--to-git`

flag whether to add data directly to Git, instead of tracking data identity only. Usually this is not desired, as it inflates dataset sizes and impacts flexibility of data transport. If not specified - it will be up to git-annex to decide, possibly on `.gitattributes` options. [Default: None]

**-nosave**

by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]

**-m MESSAGE, -message MESSAGE**

a description of the state or the changes made to a dataset. Constraints: value must be a string [Default: None]

**-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

**-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

**-S, -ds2super, -datasets-to-super**

given paths of dataset (toplevel) locations will cause these datasets to be added to their respective superdatasets underneath a given base DATASET (instead of all their content to themselves). If no base DATASET is provided, this flag has no effect. Regular files and directories are always added to their respective datasets, regardless of this setting. [Default: False]

**-git-opts STRING**

option string to be passed to git calls. Constraints: value must be a string [Default: None]

**-annex-opts STRING**

option string to be passed to git annex calls. Constraints: value must be a string [Default: None]

**-annex-add-opts STRING**

option string to be passed to git annex add calls. Constraints: value must be a string [Default: None]

**-J NJOBS, -jobs NJOBS**

how many parallel jobs (where possible) to use. Constraints: value must be convertible to type 'int' [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.



## datalad-create

### Synopsis

```
datalad-create [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-f] [-D
↳DESCRIPTION] [-d PATH] [--no-annex] [--nosave] [--annex-version ANNEX_VERSION] [--
↳annex-backend ANNEX_BACKEND] [--native-metadata-type LABEL] [--shared-access MODE]
↳[--git-opts STRING] [--annex-opts STRING] [--annex-init-opts STRING] [PATH]
```

### Description

Create a new dataset from scratch.

This command initializes a new dataset at a given location, or the current directory. The new dataset can optionally be registered in an existing superdataset (the new dataset's path needs to be located within the superdataset for that, and the superdataset needs to be given explicitly). It is recommended to provide a brief description to label the dataset's nature *and* location, e.g. "Michael's music on black laptop". This helps humans to identify data locations in distributed scenarios. By default an identifier comprised of user and machine name, plus path will be generated.

This command only creates a new dataset, it does not add any content to it, even if the target directory already contains additional files or directories.

Plain Git repositories can be created via the `--no-annex` flag. However, the result will not be a full dataset, and, consequently, not all features are supported (e.g. a description).

To create a local version of a remote dataset use the `install` command instead.

**NOTE** Power-user info: This command uses `git init`, and `git annex init` to prepare the new dataset. Registering to a superdataset is performed via a git submodule add operation in the discovered superdataset.

### Options

#### PATH

path where the dataset shall be created, directories will be created as necessary. If no location is provided, a dataset will be created in the current working directory. Either way the command will error if the target directory is not empty. Use `FORCE` to create a dataset in a non-empty directory. Constraints: value must be a string, or Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

#### `--version`

show the program's version and license information

#### `-h, --help, --help-np`

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

#### `-l LEVEL, --log-level LEVEL`

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-f, -force**

enforce creation of a dataset in a non-empty directory. [Default: False]

**-D DESCRIPTION, -description DESCRIPTION**

short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string [Default: None]

**-d PATH, -dataset PATH**

specify the dataset to perform the create operation on. If a dataset is give, a new subdataset will be created in it. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-no-annex**

if set, a plain Git repository will be created without any annex. [Default: False]

**-nosave**

by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]

**-annex-version ANNEX\_VERSION**

select a particular annex repository version. The list of supported versions depends on the available git-annex version. This should be left untouched, unless you know what you are doing. Constraints: value must be convertible to type ‘int’ [Default: None]

**-annex-backend ANNEX\_BACKEND**

set default hashing backend used by the new dataset. For a list of supported backends see the git-annex documentation. The default is optimized for maximum compatibility of datasets across platforms (especially those with limited path lengths). Constraints: value must be a string [Default: ‘MD5E’]

**-native-metadata-type LABEL**

Metadata type label. Must match the name of the respective parser implementation in Datalad (e.g. “bids”). This option can be given multiple times. Constraints: value must be a string [Default: None]

**–shared-access MODE**

configure shared access to a dataset, see *git init –shared* documentation for complete details on the supported scenarios. Possible values include: ‘false’, ‘true’, ‘group’, and ‘all’. [Default: None]

**–git-opts STRING**

option string to be passed to git calls. Constraints: value must be a string [Default: None]

**–annex-opts STRING**

option string to be passed to git annex calls. Constraints: value must be a string [Default: None]

**–annex-init-opts STRING**

option string to be passed to git annex init calls. Constraints: value must be a string [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-create-sibling****Synopsis**

```
datalad-create-sibling [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-s
↪ [NAME]] [--target-dir PATH] [--target-url URL] [--target-pushurl URL] [--dataset
↪ DATASET] [-r] [--recursion-limit LEVELS] [--existing MODE] [--shared
↪ false|true|umask|group|all|world|everybody|0xxx] [--ui false|true|html_filename] [--
↪ as-common-datasrc NAME] [--publish-by-default REFSPEC] [--publish-depends
↪ SIBLINGNAME] [--annex-wanted EXPR] [--annex-group EXPR] [--annex-groupwanted EXPR]
↪ [--inherit] [--since SINCE] [SSHURL]
```

**Description**

Create a dataset sibling on a UNIX-like SSH-accessible machine

Given a local dataset, and SSH login information this command creates a remote dataset repository and configures it as a dataset sibling to be used as a publication target (see PUBLISH command).

Various properties of the remote sibling can be configured (e.g. name location on the server, read and write access URLs, and access permissions).

Optionally, a basic web-viewer for DataLad datasets can be installed at the remote location.

This command supports recursive processing of dataset hierarchies, creating a remote sibling for each dataset in the hierarchy. By default, remote siblings are created in hierarchical structure that reflects the organization on the local file system. However, a simple templating mechanism is provided to produce a flat list of datasets (see –target-dir).

## Options

### SSHURL

Login information for the target server. This can be given as a URL (`ssh://host/path`) or SSH-style (`user@host:path`). Unless overridden, this also serves the future dataset's access URL and path on the server. Constraints: value must be a string

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-s [NAME], -name [NAME]**

sibling name to create for this publication target. If `RECURSIVE` is set, the same name will be used to label all the subdatasets' siblings. When creating a target dataset fails, no sibling is added. Constraints: value must be a string [Default: None]

### **-target-dir PATH**

path to the directory *on the server* where the dataset shall be created. By default the SSH access URL is used to identify this directory. If a relative path is provided here, it is interpreted as being relative to the user's home directory on the server. Additional features are relevant for recursive processing of datasets with subdatasets. By default, the local dataset structure is replicated on the server. However, it is possible to provide a template for generating different target directory names for all (sub)datasets. Templates can contain certain placeholder that are substituted for each (sub)dataset. For example: `"/mydirectory/dataset%RELNAME"`. Supported placeholders: `%RELNAME` - the name of the datasets, with any slashes replaced by dashes . Constraints: value must be a string [Default: None]

### **-target-url URL**

“public” access URL of the to-be-created target dataset(s) (default: SSHURL). Accessibility of this URL determines the access permissions of potential consumers of the dataset. As with `TARGET_DIR`, templates (same set of placeholders) are supported. Also, if specified, it is provided as the annex description . Constraints: value must be a string [Default: None]

**-target-pushurl URL**

In case the TARGET\_URL cannot be used to publish to the dataset, this option specifies an alternative URL for this purpose. As with TARGET\_URL, templates (same set of placeholders) are supported. . Constraints: value must be a string [Default: None]

**-dataset DATASET, -d DATASET**

specify the dataset to create the publication target for. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

**-existing MODE**

action to perform, if a sibling is already configured under the given name and/or a target directory already exists. In this case, a dataset can be skipped ('skip'), an existing target directory be forcefully re-initialized, and the sibling (re-)configured ('replace', implies 'reconfigure'), the sibling configuration be updated only ('reconfigure'), or to error ('error'). Constraints: value must be one of ('skip', 'replace', 'error', 'reconfigure') [Default: 'error']

**-shared false|true|umask|group|all|world|everybody|0xxx**

if given, configures the access permissions on the server for multi-users (this could include access by a webserver!). Possible values for this option are identical to those of *git init --shared* and are described in its documentation. Constraints: value must be a string, or value must be convertible to type bool [Default: None]

**-ui false|true|html\_filename**

publish a web interface for the dataset with an optional user-specified name for the html at publication target. defaults to INDEX.HTML at dataset root. Constraints: value must be convertible to type bool, or value must be a string [Default: False]

**-as-common-datasrc NAME**

configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto- enabled special remote). [Default: None]

### **–publish-by-default REFSPEC**

add a refspec to be published to this sibling by default if nothing specified. Constraints: value must be a string [Default: None]

### **–publish-depends SIBLINGNAME**

add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item ‘remote.SIBLINGNAME .datalad-publish-depends’. This option can be given more than once to configure multiple dependencies. Constraints: value must be a string [Default: None]

### **–annex-wanted EXPR**

expression to specify ‘wanted’ content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-wanted/> for more information. Constraints: value must be a string [Default: None]

### **–annex-group EXPR**

expression to specify a group for the repository. See <https://git-annex.branchable.com/git-annex-group/> for more information. Constraints: value must be a string [Default: None]

### **–annex-groupwanted EXPR**

expression for the groupwanted. Makes sense only if –annex-wanted=”groupwanted” and annex-group is given too. See <https://git-annex.branchable.com/git-annex-groupwanted/> for more information. Constraints: value must be a string [Default: None]

### **–inherit**

if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: False]

### **–since SINCE**

limit processing to datasets that have been changed since a given state (by tag, branch, commit, etc). This can be used to create siblings for recently added subdatasets. Constraints: value must be a string [Default: None]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## datalad-create-sibling-github

### Synopsis

```
datalad-create-sibling-github [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [--
↪dataset DATASET] [-r] [--recursion-limit LEVELS] [-s NAME] [--existing MODE] [--
↪github-login NAME] [--github-passwd PASSWORD] [--github-organization NAME] [--
↪access-protocol ACCESS_PROTOCOL] [--publish-depends SIBLINGNAME] [--dryrun] REPONAME
```

### Description

Create dataset sibling on Github.

A repository can be created under a user's Github account, or any organization a user is a member of (given appropriate permissions).

Recursive sibling creation for subdatasets is supported. A dataset hierarchy is represented as a flat list of Github repositories.

Github cannot host dataset content. However, in combination with other data sources (and siblings), publishing a dataset to Github can facilitate distribution and exchange, while still allowing any dataset consumer to obtain actual data content from alternative sources.

For Github authentication user credentials can be given as arguments. Alternatively, they are obtained interactively or queried from the systems credential store. Lastly, an *oauth* token stored in the Git configuration under variable *hub.oauthtoken* will be used automatically. Such a token can be obtained, for example, using the commandline Github interface (<https://github.com/sociomantic/git-hub>) by running: `git hub setup`.

### Options

#### REPONAME

Github repository name. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string

#### -version

show the program's version and license information

#### -h, -help, -help-np

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

#### -l LEVEL, -log-level LEVEL

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**–pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**–dataset DATASET, -d DATASET**

specify the dataset to create the publication target for. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-r, –recursive**

if set, recurse into potential subdataset. [Default: False]

**–recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**-s NAME, –name NAME**

name to represent the Github repository in the local dataset installation. Constraints: value must be a string [Default: ‘github’]

**–existing MODE**

desired behavior when already existing or configured siblings are discovered. ‘skip’: ignore; ‘error’: fail immediately; ‘reconfigure’: use the existing repository and reconfigure the local dataset to use it as a sibling. Constraints: value must be one of (‘skip’, ‘error’, ‘reconfigure’) [Default: ‘error’]

**–github-login NAME**

Github user name or access token. Constraints: value must be a string [Default: None]

**–github-passwd PASSWORD**

Github user password. Constraints: value must be a string [Default: None]

**–github-organization NAME**

If provided, the repository will be created under this Github organization. The respective Github user needs appropriate permissions. Constraints: value must be a string [Default: None]



**–access-protocol ACCESS\_PROTOCOL**

Which access protocol/URL to configure for the sibling. Constraints: value must be one of ('https', 'ssh') [Default: 'https']

**–publish-depends SIBLINGNAME**

add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME .datalad-publish-depends'. This option can be given more than once to configure multiple dependencies. Constraints: value must be a string [Default: None]

**–dryrun**

If this flag is set, no communication with Github is performed, and no repositories will be created. Instead would-be repository names are reported for all relevant datasets . [Default: False]

**Authors**

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

**datalad-drop****Synopsis**

```
datalad-drop [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-r] [--
↪ recursion-limit LEVELS] [--nocheck] [--if-dirty {fail,save-before,ignore}] [PATH_
↪ [PATH ...]]
```

**Description**

Drop file content from datasets

This command takes any number of paths of files and/or directories. If a common (super)dataset is given explicitly, the given paths are interpreted relative to this dataset.

Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

By default, the availability of at least one remote copy is verified, before file content is dropped. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

**Examples**

Drop all file content in a dataset:

```
~/some/dataset$ datalad drop
```

Drop all file content in a dataset and all its subdatasets:

```
~/some/dataset$ datalad drop --recursive
```

## Options

### PATH

path/name of the component to be dropped. Constraints: value must be a string [Default: None]

### -version

show the program's version and license information

### -h, -help, -help-np

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

### -l LEVEL, -log-level LEVEL

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### -pbs-runner {condor}

execute command by scheduling it via available PBS. For settings, config file will be consulted

### -d DATASET, -dataset DATASET

specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### -r, -recursive

if set, recurse into potential subdataset. [Default: False]

### -recursion-limit LEVELS

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

### -nocheck

whether to perform checks to assure the configured minimum number (remote) source for data. Give this option to skip checks. [Default: True]

**-if-dirty {fail,save-before,ignore}**

desired behavior if a dataset with unsaved changes is discovered: 'fail' will trigger an error and further processing is aborted; 'save-before' will save all changes prior any further action; 'ignore' let's datalad proceed as if the dataset would not have unsaved changes. [Default: 'save-before']

**Authors**

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

**datalad-export****Synopsis**

```
datalad-export [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [--  
↪help-type] [-o OUTPUT] {tarball}
```

**Description**

Export a dataset to another representation

**Options****{tarball}**

label of the type or format the dataset shall be exported to.

**-version**

show the program's version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, --dataset DATASET**

specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path)

### **--help-type**

show help for a specific export type/format. [Default: False]

### **-o OUTPUT, --output OUTPUT**

output destination specification to be passes to the exporter. The particular semantics of the option value depend on the actual exporter. Typically, this will be a file name or a path to a directory. [Default: None]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-get**

### **Synopsis**

```
datalad-get [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-s LABEL] [-d PATH]
↳ [-r] [--recursion-limit LEVELS] [-n] [-D DESCRIPTION] [--reckless] [-J NJOBS] [-v]
↳ [PATH [PATH ...]]
```

### **Description**

Get any dataset content (files/directories/subdatasets).

This command only operates on dataset content. To obtain a new independent dataset from some source use the `INSTALL` command.

By default this command operates recursively within a dataset, but not across potential subdatasets, i.e. if a directory is provided, all files in the directory are obtained. Recursion into subdatasets is supported too. If enabled, relevant subdatasets are detected and installed in order to fulfill a request.

Known data locations for each requested file are evaluated and data are obtained from some available location (according to git-annex configuration and possibly assigned remote priorities), unless a specific source is specified.

**NOTE** Power-user info: This command uses git annex get to fulfill file handles.

### **Options**

#### **PATH**

path/name of the requested dataset component. The component must already be known to a dataset. To add new components to a dataset use the `ADD` command. Constraints: value must be a string [Default: None]

**-version**

show the program's version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-s LABEL, -source LABEL**

label of the data source to be used to fulfill requests. This can be the name of a dataset sibling or another known source. Constraints: value must be a string [Default: None]

**-d PATH, -dataset PATH**

specify the dataset to perform the add operation on, in which case `PATH` arguments are interpreted as being relative to this dataset. If no dataset is given, an attempt is made to identify a dataset for each input `PATH`. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

**-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Alternatively, 'existing' will limit recursion to subdatasets that already existed on the filesystem at the start of processing, and prevent new subdatasets from being obtained recursively. Constraints: value must be convertible to type 'int', or value must be one of ('existing',) [Default: None]

**-n, -no-data**

whether to obtain data for all file handles. If disabled, GET operations are limited to dataset handles. This option prevents data for file handles from being obtained. [Default: True]

### **-D DESCRIPTION, --description DESCRIPTION**

short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string [Default: None]

### **--reckless**

Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: False]

### **-J NJOBS, --jobs NJOBS**

how many parallel jobs (where possible) to use. Constraints: value must be convertible to type ‘int’ [Default: None]

### **-v, --verbose**

print out more detailed information while executing a command. [Default: False]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-install**

### **Synopsis**

```
datalad-install [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-s SOURCE] [-d DATASET] [-g] [-D DESCRIPTION] [-r] [--recursion-limit LEVELS] [--nosave] [--reckless] [-J NJOBS] [PATH [PATH ...]]
```

### **Description**

Install a dataset from a (remote) source.

This command creates a local sibling of an existing dataset from a (remote) location identified via a URL or path. Optional recursion into potential subdatasets, and download of all referenced data is supported. The new dataset can be optionally registered in an existing superdataset by identifying it via the DATASET argument (the new dataset’s path needs to be located within the superdataset for that).

It is recommended to provide a brief description to label the dataset’s nature *and* location, e.g. “Michael’s music on black laptop”. This helps humans to identify data locations in distributed scenarios. By default an identifier comprised of user and machine name, plus path will be generated.

When only partial dataset content shall be obtained, it is recommended to use this command without the GET-DATA flag, followed by a *get* operation to obtain the desired data.

**NOTE** Power-user info: This command uses `git clone`, and `git annex init` to prepare the dataset. Registering to a superdataset is performed via a `git submodule add` operation in the discovered superdataset.

## Options

### PATH

path/name of the installation target. If no PATH is provided a destination path will be derived from a source URL similar to `git clone`. [Default: None]

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among `critical`, `error`, `warning`, `info`, `debug`. Also you can specify an integer `<10` to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-s SOURCE, -source SOURCE**

URL or local path of the installation source. Constraints: value must be a string [Default: None]

### **-d DATASET, -dataset DATASET**

specify the dataset to perform the install operation on. If no dataset is given, an attempt is made to identify the dataset in a parent directory of the current working directory and/or the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-g, -get-data**

if given, obtain all data content too. [Default: False]

**-D DESCRIPTION, --description DESCRIPTION**

short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string [Default: None]

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**--recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**--nosave**

by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]

**--reckless**

Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: False]

**-J NJOBS, --jobs NJOBS**

how many parallel jobs (where possible) to use. Constraints: value must be convertible to type ‘int’ [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-publish**

**Synopsis**

```
datalad-publish [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [--  
→to LABEL] [--since SINCE] [--missing MODE] [-f] [-r] [--recursion-limit LEVELS] [--  
→git-opts STRING] [--annex-opts STRING] [--annex-copy-opts STRING] [PATH [PATH ...]]
```



## Description

Publish a dataset to a known sibling.

This makes the last saved state of a dataset available to a sibling or special remote data store of a dataset. Any target sibling must already exist and be known to the dataset.

Optionally, it is possible to limit publication to change sets relative to a particular point in the version history of a dataset (e.g. a release tag). By default, the state of the local dataset is evaluated against the last known state of the target sibling. Actual publication is only attempted if there was a change compared to the reference state, in order to speed up processing of large collections of datasets. Evaluation with respect to a particular “historic” state is only supported in conjunction with a specified reference dataset. Change sets are also evaluated recursively, i.e. only those subdatasets are published where a change was recorded that is reflected in to current state of the top-level reference dataset. See “since” option for more information.

Only publication of saved changes is supported. Any unsaved changes in a dataset (hierarchy) have to be saved before publication.

**NOTE** Power-user info: This command uses git push, and git annex copy to publish a dataset. Publication targets are either configured remote Git repositories, or git-annex special remotes (if their support data upload).

## Options

### PATH

path(s), that may point to file handle(s) to publish including their actual content or to subdataset(s) to be published. If a file handle is published with its data, this implicitly means to also publish the (sub)dataset it belongs to. ‘.’ as a path is treated in a special way in the sense, that it is passed to subdatasets in case RECURSIVE is also given. Constraints: value must be a string [Default: None]

### -version

show the program’s version and license information

### -h, -help, -help-np

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

### -l LEVEL, -log-level LEVEL

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### -pbs-runner {condor}

execute command by scheduling it via available PBS. For settings, config file will be consulted

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

specify the (top-level) dataset to be published. If no dataset is given, the datasets are determined based on the input arguments. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-to LABEL**

name of the target sibling. If no name is given an attempt is made to identify the target based on the dataset's configuration (i.e. a configured tracking branch, or a single sibling that is configured for publication). Constraints: value must be a string [Default: None]

**--since SINCE**

When publishing dataset(s), specifies commit (treeish, tag, etc) from which to look for changes to decide either updated publishing is necessary for this and which children. If empty argument is provided, then we will always run publish command. By default, would take from the previously published to that remote/sibling state (for the current branch). Constraints: value must be a string [Default: None]

**--missing MODE**

action to perform, if a sibling does not exist in a given dataset. By default it would fail the run ('fail' setting). With 'inherit' a 'create-sibling' with '--inherit-settings' will be used to create sibling on the remote. With 'skip' - it simply will be skipped. Constraints: value must be one of ('fail', 'inherit', 'skip') [Default: 'fail']

**-f, --force**

enforce doing publish activities (git push etc) regardless of the analysis if they seemed needed. [Default: False]

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**--recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

**--git-opts STRING**

option string to be passed to git calls. Constraints: value must be a string [Default: None]

**--annex-opts STRING**

option string to be passed to git annex calls. Constraints: value must be a string [Default: None]

**-annex-copy-opts STRING**

option string to be passed to git annex copy calls. Constraints: value must be a string [Default: None]

**Authors**

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

**datalad-remove****Synopsis**

```
datalad-remove [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-r]
↳ [--nocheck] [--if-dirty {fail,save-before,ignore}] [PATH [PATH ...]]
```

**Description**

Remove components from datasets

This command can remove any components (subdatasets, and (directories with) files) from datasets. Removing a component implies any present content to be dropped, and any associated subdatasets to be uninstalled. Subsequently, the component is “unregistered” from the respective dataset. This means that the respective component is no longer present on the file system.

By default, the availability of at least one remote copy is verified, by default, before file content is dropped. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

Any number of paths to process can be given as input. Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

**Examples**

Permanently remove a subdataset from a dataset and wipe out the subdataset association too:

```
~/some/dataset$ datalad remove somesubdataset1
```

**Options****PATH**

path/name of the component to be removed. Constraints: value must be a string [Default: None]

**-version**

show the program’s version and license information

### **-h, --help, --help-np**

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

### **-l LEVEL, --log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, --dataset DATASET**

specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

### **--nocheck**

whether to perform checks to assure the configured minimum number (remote) source for data. Give this option to skip checks. [Default: True]

### **--if-dirty {fail,save-before,ignore}**

desired behavior if a dataset with unsaved changes is discovered: 'fail' will trigger an error and further processing is aborted; 'save-before' will save all changes prior any further action; 'ignore' let's datalad proceed as if the dataset would not have unsaved changes. [Default: 'save-before']

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-save**

### **Synopsis**

```
datalad-save [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-m MESSAGE] [-d DATASET] [-u] [-a] [--version-tag ID] [-r] [--recursion-limit LEVELS] [-S] [FILES ...]
```

## Description

Save the current state of a dataset

Saving the state of a dataset records all changes that have been made to it. This change record is annotated with a user-provided description. Optionally, an additional tag, such as a version, can be assigned to the saved state. Such tag enables straightforward retrieval of past versions at a later point in time.

## Options

### FILES

list of files to consider. If given, only changes made to those files are recorded in the new state. Constraints: value must be a string [Default: None]

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-m MESSAGE, -message MESSAGE**

a description of the state or the changes made to a dataset. Constraints: value must be a string [Default: None]

### **-d DATASET, -dataset DATASET**

“specify the dataset to save. If a dataset is given, but no FILES, the entire dataset will be saved. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-u, -all-updated**

if no explicit paths are given, save changes of all known components in a datasets. [Default: True]

### **-a, --all-changes**

save all changes (even to not yet added files) of all components in datasets that contain any of the given paths [DEPRECATED!]. [Default: None]

### **--version-tag ID**

an additional marker for that state. Constraints: value must be a string [Default: None]

### **-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

### **--recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

### **-S, --super-datasets**

if set, save a change in a dataset also in its superdataset. [Default: False]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-update**

### **Synopsis**

```
datalad-update [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-s SIBLING] [--merge] [-d DATASET] [-r] [--recursion-limit LEVELS] [--fetch-all] [--reobtain-data] [PATH [PATH ...]]
```

### **Description**

Update a dataset from a sibling.

### **Options**

#### **PATH**

path to be updated. Constraints: value must be a string [Default: None]

**-version**

show the program's version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-s *SIBLING*, -sibling *SIBLING***

name of the sibling to update from. Constraints: value must be a string [Default: None]

**-merge**

merge obtained changes from the given or the default sibling. [Default: False]

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

“specify the dataset to update. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

**-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**-fetch-all**

fetch updates from all known siblings. [Default: False]

### **-reobtain-data**

TODO. [Default: False]

### **Authors**

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

### **datalad-uninstall**

#### **Synopsis**

```
datalad-uninstall [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-r] [--nocheck] [--if-dirty {fail,save-before,ignore}] [PATH [PATH ...]]
```

#### **Description**

Uninstall subdatasets

This command can be used to uninstall any number of installed subdataset. If a to-be-uninstalled subdataset contains presently installed subdatasets itself, their recursive removal has to be enabled explicitly to avoid the command to exit with an error. This command will error if individual files or non-dataset directories are given as input (use the drop or remove command depending in the desired goal), nor will it uninstall top-level datasets (i.e. datasets that or not a subdataset in another dataset; use the remove command for this purpose).

By default, the availability of at least one remote copy for each currently available file in any dataset is verified. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

Any number of paths to process can be given as input. Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

#### **Examples**

Uninstall a subdataset (undo installation):

```
~/some/dataset$ datalad uninstall somesubdataset1
```

#### **Options**

##### **PATH**

path/name of the component to be uninstalled. Constraints: value must be a string [Default: None]

##### **-version**

show the program's version and license information



**-h, --help, --help-np**

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

**-l LEVEL, --log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

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

specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**--nocheck**

whether to perform checks to assure the configured minimum number (remote) source for data. Give this option to skip checks. [Default: True]

**--if-dirty {fail,save-before,ignore}**

desired behavior if a dataset with unsaved changes is discovered: 'fail' will trigger an error and further processing is aborted; 'save-before' will save all changes prior any further action; 'ignore' let's datalad proceed as if the dataset would not have unsaved changes. [Default: 'save-before']

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-unlock****Synopsis**

```
datalad-unlock [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-r]
↳ [--recursion-limit LEVELS] [path [path ...]]
```

## Description

Unlock file(s) of a dataset

Unlock files of a dataset in order to be able to edit the actual content

## Options

### **path**

file(s) to unlock. Constraints: value must be a string [Default: None]

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, -dataset DATASET**

“specify the dataset to unlock files in. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. If the latter fails, an attempt is made to identify the dataset based on `PATH` . Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

### **-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

## Authors

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

## Meta data handling

### datalad-search

#### Synopsis

```
datalad-search [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-s_
↵PROPERTY] [-r PROPERTY] [-R] [-f FORMAT] [--regex] STRING [STRING ...]
```

#### Description

Search within available in datasets' meta data

#### Yields

**location** [str] (relative) path to the dataset

**report** [dict] fields which were requested by REPORT option

#### Options

##### STRING

a string (or a regular expression if `--regex`) to search for in all meta data values. If multiple provided, all must have a match among some fields of a dataset.

##### `--version`

show the program's version and license information

##### `-h, --help, --help-np`

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

##### `-l LEVEL, --log-level LEVEL`

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-d DATASET, -dataset DATASET**

specify the dataset to perform the query operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-s PROPERTY, -search PROPERTY**

name of the property to search for any match. This option can be given multiple times. By default, all properties are searched. [Default: None]

**-r PROPERTY, -report PROPERTY**

name of the property to report for any match. This option can be given multiple times. If '\*' is given, all properties are reported. [Default: None]

**-R, -report-matched**

flag to report those fields which have matches. If REPORT option values are provided, union of matched and those in REPORT will be output. [Default: False]

**-f FORMAT, -format FORMAT**

format for output. Constraints: value must be one of ('custom', 'json', 'yaml') [Default: 'custom']

**-regex**

flag for STRING to be used as a (Python) regular expression which should match the value. [Default: False]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-metadata**

**Synopsis**

```
datalad-metadata [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-a  
→KEY [VAL ...]] [-i KEY [VAL ...]] [--remove KEY [VAL ...]] [--reset KEY [VAL ...]]  
→[--define-key KEY DEFINITION] [-g] [-r] [--recursion-limit LEVELS] [PATH [PATH ...]]
```

## Description

Metadata manipulation for files and whole datasets

Two types of metadata are supported:

1. metadata describing a dataset as a whole (dataset-global), and
2. metadata for individual files in a dataset.

Both types can be accessed and modified with this command. Note, however, that this only refers to Datalad's native metadata, and not to any other metadata that is possibly stored in files of a dataset.

Datalad's native metadata capability is primarily targeting data description via arbitrary tags and other (brief) key-value attributes (with possibly multiple values for a single key).

Metadata key names are limited to alphanumeric characters (and [\_-]). Moreover, all key names are converted to lower case.

### *Dataset (global) metadata*

Metadata describing a dataset as a whole is stored in JSON format in the dataset at `.datalad/metadata/dataset.json`. The amount of metadata that can be stored is not limited by Datalad. However, it should be kept brief as this information is stored in the Git history of the dataset, and access or modification requires to read the entire file.

Arbitrary metadata keys can be used. However, Datalad reserves the keys 'tag' and 'definition' for its own use. The can still be manipulated without any restrictions like any other metadata items, but doing so can impact Datalad's metadata-related functionality, handle with care.

The 'tag' key is used to store a list of (unique) tags.

The 'definition' key is used to store key-value mappings that define metadata keys used elsewhere in the metadata. Using the feature is optional (see `--define-key`). It can be useful in the context of data discovery needs, where metadata keys can be precisely defined by linking them to specific ontology terms.

### *File metadata*

Metadata storage for individual files is provided by git-annex, and generally the same rules as for dataset-global metadata apply. However, there is just one reserved key name: 'tag'.

Again, the amount of metadata is not limited, but metadata is stored in git-annex' internal data structures in the Git repository of a dataset. Large amounts of metadata can slow its performance.

### *Output rendering*

By default, a short summary of the metadata for each dataset (component) is rendered:

```
<path> (<type>): -|<keys> [<tags>]
```

where `<path>` is the path of the respective component, `<type>` a label for the type of dataset components metadata is presented for. Non-existent metadata is indicated by a dash, otherwise a comma-separated list of metadata keys (except for 'tag'), is followed by a list of tags, if there are any.

## Options

### PATH

path(s) to set/get metadata. Constraints: value must be a string [Default: None]

**-version**

show the program's version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-d DATASET, -dataset DATASET**

Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**-a KEY [VAL ...], -add KEY [VAL ...]**

metadata items to add. If only a key is given, a corresponding tag is added. If a key-value mapping (multiple values at once are supported) is given, the values are added to the metadata item of that key. Constraints: value must be a string [Default: None]

**-i KEY [VAL ...], -init KEY [VAL ...]**

like `-add`, but tags are only added if no tag was present before. Likewise, values are only added to a metadata key, if that key did not exist before. Constraints: value must be a string [Default: None]

**-remove KEY [VAL ...]**

metadata values to remove. If only a key is given, a corresponding tag is removed. If a key-value mapping (multiple values at once are supported) is given, only those values are removed from the metadata item of that key. If no values are left after the removal, the entire item of that key is removed. Constraints: value must be a string [Default: None]

**-reset KEY [VAL ...]**

metadata items to remove. If only a key is given, a corresponding metadata key with all its values is removed. If a key-value mapping (multiple values at once are supported) is given, any existing values for this key are replaced by the given ones. Constraints: value must be a string [Default: None]

**–define-key KEY DEFINITION**

convenience option to add an item in the dataset’s global metadata (‘definition’ key). This can be used to define (custom) keys used in the datasets’s metadata, for example by providing a URL to an ontology term for a given key label. This option does not need `–dataset-global` to be set to be in effect. Constraints: value must be a string [Default: None]

**-g, –dataset-global**

Whether to perform metadata query or modification on the global dataset metadata, or on individual dataset components. For example, without this switch setting metadata using the root path of a dataset, will set the given metadata for all files in a dataset, whereas with this flag only the metadata record of the dataset itself will be altered. [Default: False]

**-r, –recursive**

if set, recurse into potential subdataset. [Default: False]

**–recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-aggregate-metadata****Synopsis**

```
datalad-aggregate-metadata [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET]
  [--guess-native-type] [-r] [--recursion-limit LEVELS] [--nosave] [--if-dirty {fail,save-before,ignore}]
```

**Description**

Aggregate meta data of a dataset for later query.

By default meta data is aggregated across all configured native meta data sources. Optionally, the type of available meta data can be guessed, if no types are configured. Moreover, it is possible to aggregate meta data from any subdatasets into the superdataset, in order to facilitate data discovery without having to obtain any subdataset.

**Returns**

**List** Any datasets where (updated) aggregated meta data was saved.

## Options

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, -dataset DATASET**

specify the dataset to perform the install operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the PATH given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path)

### **-guess-native-type**

guess native meta data type of datasets, if none is configured. With a configured, or auto-detected meta data type, no native meta data will be aggregated. [Default: False]

### **-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

### **-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

### **-nosave**

by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]



**-if-dirty {fail,save-before,ignore}**

desired behavior if a dataset with unsaved changes is discovered: ‘fail’ will trigger an error and further processing is aborted; ‘save-before’ will save all changes prior any further action; ‘ignore’ let’s datalad proceed as if the dataset would not have unsaved changes. [Default: ‘save-before’]

**Authors**

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

**Miscellaneous commands****datalad-add-archive-content****Synopsis**

```
datalad-add-archive-content [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [--
↪annex ANNEX] [--add-archive-leading-dir] [--strip-leading-dirs] [--leading-dirs-
↪depth LEADING_DIRS_DEPTH] [--leading-dirs-consider LEADING_DIRS_CONSIDER] [--use-
↪current-dir] [-d] [--key] [-e EXCLUDE] [-r RENAME] [--existing {fail,overwrite,
↪archive-suffix,numeric-suffix}] [-o ANNEX_OPTIONS] [--copy] [--no-commit] [--allow-
↪dirty] [--stats STATS] [--drop-after] [--delete-after] archive
```

**Description**

Add content of an archive under git annex control.

This results in the files within archive (which should be under annex control itself) added under annex referencing original archive via custom special remotes mechanism

Example:

```
annex-repo$ datalad add-archive-content my_big_tarball.tar.gz
```

**Options****archive**

archive file or a key (if `-key` specified). Constraints: value must be a string

**-version**

show the program’s version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-annex ANNEX**

annex instance to use. [Default: None]

**-add-archive-leading-dir**

flag to place extracted content under a directory which would correspond to archive name with suffix stripped. E.g. for archive EXAMPLE.ZIP its content will be extracted under a directory EXAMPLE/. [Default: False]

**-strip-leading-dirs**

flag to move all files directories up, from how they were stored in an archive, if that one contained a number (possibly more than 1 down) single leading directories. [Default: False]

**-leading-dirs-depth LEADING\_DIRS\_DEPTH**

maximal depth to strip leading directories to. If not specified (None), no limit. [Default: None]

**-leading-dirs-consider LEADING\_DIRS\_CONSIDER**

regular expression(s) for directories to consider to strip away. Constraints: value must be a string [Default: None]

**-use-current-dir**

flag to extract archive under the current directory, not the directory where archive is located. Note that it will be of no effect if -key is given. [Default: False]

**-d, -delete**

flag to delete original archive from the filesystem/git in current tree. Note that it will be of no effect if -key is given. [Default: False]

**-key**

flag to signal if provided archive is not actually a filename on its own but an annex key. [Default: False]

**-e EXCLUDE, --exclude EXCLUDE**

regular expressions for filenames which to exclude from being added to annex. Applied after `--rename` if that one is specified. For exact matching, use anchoring. Constraints: value must be a string [Default: None]

**-r RENAME, --rename RENAME**

regular expressions to rename files before being added under git. First letter defines how to split provided string into two parts: Python regular expression (with groups), and replacement string. Constraints: value must be a string [Default: None]

**--existing {fail,overwrite,archive-suffix,numeric-suffix}**

what operation to perform a file from archive tries to overwrite an existing file with the same name. 'fail' (default) leads to RuntimeError exception. 'overwrite' silently replaces existing file. 'archive-suffix' instructs to add a suffix (prefixed with a '-') matching archive name from which file gets extracted, and if that one present, 'numeric-suffix' is in effect in addition, when incremental numeric suffix (prefixed with a '.') is added until no name collision is longer detected. [Default: 'fail']

**-o ANNEX\_OPTIONS, --annex-options ANNEX\_OPTIONS**

additional options to pass to git-annex. Constraints: value must be a string [Default: None]

**--copy**

flag to copy the content of the archive instead of moving. [Default: False]

**--no-commit**

flag to not commit upon completion. [Default: True]

**--allow-dirty**

flag that operating on a dirty repository (uncommitted or untracked content) is ok. [Default: False]

**--stats STATS**

ActivityStats instance for global tracking. [Default: None]

**--drop-after**

drop extracted files after adding to annex. [Default: False]

### **-delete-after**

extract under a temporary directory, git-annex add, and delete after. To be used to “index” files within annex without actually creating corresponding files under git. Note that *annex dropunused* would later remove that load. [Default: False]

### **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

### **datalad-clean**

#### **Synopsis**

```
datalad-clean [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [--  
↪what [{cached-archives, annex-tmp} [{cached-archives, annex-tmp} ...]]] [-r] [--  
↪recursion-limit LEVELS]
```

#### **Description**

Clean up after DataLad (possible temporary files etc.)

Removes extracted temporary archives, etc.

Examples:

```
$ datalad clean
```

#### **Options**

##### **-version**

show the program’s version and license information

##### **-h, -help, -help-np**

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

##### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

##### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

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

specify the dataset to perform the clean operation on. If no dataset is given, an attempt is made to identify the dataset in current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**--what [{cached-archives,annex-tmp} [{cached-archives,annex-tmp} ...]]**

What to clean. If none specified – all known targets are cleaned. [Default: None]

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**--recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-crawl****Synopsis**

```
datalad-crawl [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [--is-pipeline] [-t] [-r] [-C CHDIR] [file]
```

**Description**

Crawl online resource to create or update a dataset.

Examples:

```
$ datalad crawl # within a dataset having .datalad/crawl/crawl.cfg
```

**Options****file**

configuration (or pipeline if `--is-pipeline`) file defining crawling, or a directory of a dataset on which to perform crawling using its standard crawling specification. Constraints: value must be a string [Default: None]

**-version**

show the program's version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-is-pipeline**

flag if provided file is a Python script which defines `pipeline()`. [Default: False]

**-t, -is-template**

flag if provided value is the name of the template to use. [Default: False]

**-r, -recursive**

flag to crawl subdatasets as well (for now serially). [Default: False]

**-C CHDIR, -chdir CHDIR**

directory to `chdir` to for crawling. Constraints: value must be a string [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-crawl-init**

**Synopsis**

```
datalad-crawl-init [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-t TEMPLATE] ↵  
↪ [-f TEMPLATE_FUNC] [--save] [args [args ...]]
```

## Description

Initialize crawling configuration

Allows to specify template and function to generate a crawling pipeline

Examples:

```
$ datalad crawl-init --template openfmri --template-func superdataset_pipeline
```

```
$ datalad crawl-init --template fcptable dataset=Baltimore tarballs=True
```

## Options

### args

keyword arguments to pass into the template function generating actual pipeline, organized in key=value pairs. Constraints: value must be a string [Default: None]

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-t TEMPLATE, --template TEMPLATE**

the name of the template. Constraints: value must be a string [Default: None]

### **-f TEMPLATE\_FUNC, --template-func TEMPLATE\_FUNC**

the name of the function. [Default: None]

### **-save**

flag to save file into git repo. [Default: False]

### Authors

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

### datalad-download-url

#### Synopsis

```
datalad-download-url [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-O PATH] [-o] [-x] url [url ...]
```

#### Description

Download content

It allows for a uniform download interface to various supported URL schemes, re-using or asking for authentication detail maintained by datalad.

Examples:

```
$ datalad download http://example.com/file.dat s3://bucket/file2.dat
```

#### Options

##### url

URL(s) to be downloaded. Constraints: value must be a string

##### -version

show the program's version and license information

##### -h, -help, -help-np

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

##### -l LEVEL, -log-level LEVEL

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

##### --pbs-runner {condor}

execute command by scheduling it via available PBS. For settings, config file will be consulted



**-O PATH, --path PATH**

path (filename or directory path) where to store downloaded file(s). In case of multiple URLs provided, must point to a directory. Otherwise current directory is used. Constraints: value must be a string [Default: None]

**-o, --overwrite**

flag to overwrite it if target file exists. [Default: False]

**-x, --stop-on-failure**

flag to stop subsequent downloads upon first failure to download. [Default: False]

**Authors**

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

**datalad-ls****Synopsis**

```
datalad-ls [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-r] [-F] [-a] [-L] [--
↪config-file CONFIG_FILE] [--list-content {None,first10,md5,full}] [--json {file,
↪display,delete}] [PATH/URL [PATH/URL ...]]
```

**Description**

List summary information about URLs and dataset(s)

ATM only s3:// URLs and datasets are supported

**Examples**

```
$ datalad ls s3://openfmri/tarballs/ds202 # to list S3 bucket $ datalad ls # to list current dataset
```

**Options****PATH/URL**

URL or path to list, e.g. s3://... Constraints: value must be a string

**--version**

show the program's version and license information

**-h, --help, --help-np**

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

**-l LEVEL, --log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

**-r, --recursive**

recurse into subdirectories. [Default: False]

**-F, --fast**

only perform fast operations. Would be overridden by --all. [Default: False]

**-a, --all**

list all (versions of) entries, not e.g. only latest entries in case of S3. [Default: False]

**-L, --long**

list more information on entries (e.g. acl, urls in s3, annex sizes etc). [Default: False]

**--config-file CONFIG\_FILE**

path to config file which could help the 'ls'. E.g. for s3:// URLs could be some ~/.s3cfg file which would provide credentials. Constraints: value must be a string [Default: None]

**--list-content {None,first10,md5,full}**

list also the content or only first 10 bytes (first10), or md5 checksum of an entry. Might require expensive transfer and dump binary output to your screen. Do not enable unless you know what you are after. [Default: False]

**--json {file,display,delete}**

metadata json of dataset for creating web user interface. display: prints jsons to stdout or file: writes each subdir metadata to json file in subdir of dataset or delete: deletes all metadata json files in dataset. [Default: None]

## Authors

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

## datalad-test

### Synopsis

```
datalad-test [--version] [-h] [-l LEVEL] [--pbs-runner {condor}]
```

### Description

Run internal DataLad (unit)tests.

This can be used to verify correct operation on the system

### Options

#### **-version**

show the program's version and license information

#### **-h, -help, -help-np**

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

#### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

#### **--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

## Authors

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

## Plumbing commands

### datalad-annotate-paths

## Synopsis

```
datalad-annotate-paths [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-r] [--recursion-limit LEVELS] [--action LABEL] [--unavailable-path-status LABEL] [--unavailable-path-msg message] [--nondataset-path-status LABEL] [--no-parentds-discovery] [--no-subds-discovery] [--modified [MODIFIED]] [PATH [PATH ..]]
```

## Description

Analyze and act upon input paths

Given paths (or more generally location requests) are inspected and annotated with a number of properties. A list of recognized properties is provided below.

*Recognized path properties*

“**action**” label of the action that triggered the path annotation

“**annexkey**” annex key for the content of a file

“**logger**” logger for reporting a message

“**message**” message (plus possible tstring expansion arguments)

“**orig\_request**” original input by which a path was determined

“**parentds**” path of dataset containing the annotated path (superdataset for subdatasets)

“**path**” absolute path that is annotated

“**process\_content**” flag that content underneath the path is to be processed

“**process\_updated\_only**” flag that only known dataset components are to be processed

“**raw\_input**” flag whether this path was given as raw (non-annotated) input

“**refds**” path of a reference/base dataset the annotated path is part of

“**registered\_subds**” flag whether a dataset is known to be a true subdataset of PARENTDS

“**revision**” the recorded commit for a subdataset in a superdataset

“**revision\_descr**” a human-readable description of REVISION

“**source\_url**” URL a dataset was installed from

“**staged**” flag whether a path is known to be “staged” in its containing dataset

“**state**” state indicator for a path in its containing dataset (clean, modified, absent (also for files), conflict)

“**status**” action result status (ok, notneeded, impossible, error)

“**type**” nature of the path (file, directory, dataset)

“**url**” registered URL for a subdataset in a superdataset

In the case of enabled modification detection the results may contain additional properties regarding the nature of the modification. See the documentation of the DIFF command for details.

## Options

### PATH

path to be annotated. Constraints: value must be a string [Default: None]

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, -dataset DATASET**

an optional reference/base dataset for the paths. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-r, -recursive**

if set, recurse into potential subdataset. [Default: False]

### **-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int' [Default: None]

### **-action LABEL**

an "action" property value to include in the path annotation. Constraints: value must be a string [Default: None]

### **-unavailable-path-status LABEL**

a "status" property value to include in the annotation for paths that are underneath a dataset, but do not exist on the filesystem. Constraints: value must be a string [Default: ""]

### **–unavailable-path-msg message**

a “message” property value to include in the annotation for paths that are underneath a dataset, but do not exist on the filesystem. Constraints: value must be a string [Default: None]

### **–nondataset-path-status LABEL**

a “status” property value to include in the annotation for paths that are not underneath any dataset. Constraints: value must be a string [Default: ‘error’]

### **–no-parentds-discovery**

Flag to disable reports of parent dataset information for any path, in particular dataset root paths. Disabling saves on command run time, if this information is not needed. [Default: True]

### **–no-subds-discovery**

Flag to disable reporting type=‘dataset’ for subdatasets, even when they are not installed, or their mount point directory doesn’t exist. Disabling saves on command run time, if this information is not needed. [Default: True]

### **–modified [*MODIFIED*]**

comparison reference specification for modification detection. This can be (mostly) anything that *git diff* understands (commit, treeish, tag, etc). See the documentation of *datalad diff –revision* for details. Unmodified paths will not be annotated. If a requested path was not modified but some content underneath it was, then the request is replaced by the modified paths and those are annotated instead. This option can be used without an argument to test against changes that have been made, but have not yet been staged for a commit. Constraints: value must be a string, or value must be convertible to type bool [Default: None]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-clone**

### **Synopsis**

```
datalad-clone [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-D DESCRIPTION] [--reckless] [--alternative-sources SOURCE [SOURCE ...]] SOURCE [PATH]
```

### **Description**

Obtain a dataset copy from a URL or local source (path)

The purpose of this command is to obtain a new clone (copy) of a dataset and place it into a not-yet-existing or empty directory. As such CLONE provides a strict subset of the functionality offered by INSTALL. Only a single dataset can

be obtained, recursion is not supported. However, once installed, arbitrary dataset components can be obtained via a subsequent GET command.

Primary differences over a direct *git clone* call are 1) the automatic initialization of a dataset annex (pure Git repositories are equally supported); 2) automatic registration of the newly obtained dataset as a subdataset (submodule), if a parent dataset is specified; 3) support for datalad's resource identifiers and automatic generation of alternative access URL for common cases (such as appending '.git' to the URL in case the accessing the base URL failed); and 4) ability to take additional alternative source locations as an argument.

## Options

### SOURCE

URL, Datalad resource identifier, local path or instance of dataset to be cloned. Constraints: value must be a string

### PATH

path to clone into. If no PATH is provided a destination path will be derived from a source URL similar to git clone. [Default: None]

### -version

show the program's version and license information

### -h, -help, -help-np

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

### -l LEVEL, -log-level LEVEL

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### -pbs-runner {condor}

execute command by scheduling it via available PBS. For settings, config file will be consulted

### -d DATASET, -dataset DATASET

(parent) dataset to clone into. If given, the newly cloned dataset is registered as a subdataset of the parent. Also, if given, relative paths are interpreted as being relative to the parent dataset, and not relative to the working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-D DESCRIPTION, -description DESCRIPTION**

short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string [Default: None]

### **-reckless**

Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: False]

### **-alternative-sources SOURCE [SOURCE ...]**

Alternative sources to be tried if a dataset cannot be obtained from the main SOURCE. Constraints: value must be a string [Default: None]

## **Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

## **datalad-create-test-dataset**

### **Synopsis**

```
datalad-create-test-dataset [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [--  
↪spec SPEC] [--seed SEED] path
```

### **Description**

Create test (meta-)dataset.

### **Options**

#### **path**

path/name where to create (if specified, must not exist). Constraints: value must be a string [Default: None]

#### **-version**

show the program’s version and license information

#### **-h, -help, -help-np**

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



### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-spec SPEC**

spec for hierarchy, defined as a min-max (min could be omitted to assume 0) defining how many (random number from min to max) of sub-datasets to generate at any given level of the hierarchy. Each level separated from each other with /. Example: 1-3/-2 would generate from 1 to 3 subdatasets at the top level, and up to two within those at the 2nd level . Constraints: value must be a string [Default: None]

### **-seed SEED**

seed for rng. Constraints: value must be convertible to type 'int' [Default: None]

## **Authors**

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

## **datalad-diff**

### **Synopsis**

```
datalad-diff [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [--
↪revision [REVISION EXPRESSION]] [--staged] [--ignore-subdatasets IGNORE_
↪SUBDATASETS] [-r] [--recursion-limit LEVELS] [PATH [PATH ...]]
```

### **Description**

Report changes of dataset component between revisions.

### **Options**

#### **PATH**

path to be evaluated. Constraints: value must be a string [Default: None]

#### **-version**

show the program's version and license information

**-h, --help, --help-np**

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

**-l LEVEL, --log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

**--pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

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

specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

**--revision [REVISION EXPRESSION]**

comparison reference specification. Three modes are supported: 1) <revision> changes you have in your working tree relative to the named revision (this can also be a branch name, tag, commit or any label Git can understand). 2) <revision>..<revision> changes between two arbitrary revisions. 3) <revision>...<revision> changes on the branch containing and up to the second <revision>, starting at a common ancestor of both revisions. [Default: 'HEAD']

**--staged**

get the changes already staged for a commit relative to an optionally given revision (by default the most recent one). [Default: False]

**--ignore-subdatasets IGNORE\_SUBDATASETS**

speed up execution by (partially) not evaluating the state of subdatasets in a parent dataset. With “none” a subdataset is considered modified when it either contains untracked or modified content or its last saved state differs from that recorded in the parent dataset. When “untracked” is used subdatasets are not considered modified when they only contain untracked content (but they are still scanned for modified content). Using “dirty” ignores all changes to the work tree of subdatasets, only changes to the revisions stored in the parent dataset are shown. Using “all” hides all changes to subdatasets. Note, even with “all” recursive execution will still report other changes in any existing subdataset, only the subdataset record in a parent dataset is not evaluated. Constraints: value must be one of ('none', 'untracked', 'dirty', 'all') [Default: 'none']

**-r, --recursive**

if set, recurse into potential subdataset. [Default: False]

**-recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**Authors**

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

**datalad-sshrun****Synopsis**

```
datalad-sshrun [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-p PORT] [-n]
→login cmd
```

**Description**

Run command on remote machines via SSH.

This is a replacement for a small part of the functionality of SSH. In addition to SSH alone, this command can make use of datalad’s SSH connection management. Its primary use case is to be used with Git as ‘core.sshCommand’ or via “GIT\_SSH\_COMMAND”.

**Options****login**

[user@]hostname.

**cmd**

command for remote execution.

**-version**

show the program’s version and license information

**-h, -help, -help-np**

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

**-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-p *PORT*, -port *PORT***

port to connect to on the remote host. [Default: None]

### **-n**

Redirect stdin from /dev/null. [Default: False]

## **Authors**

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

## **datalad-siblings**

### **Synopsis**

```
datalad-siblings [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET] [-s 
↳NAME] [--url [URL]] [--pushurl PUSHURL] [-D DESCRIPTION] [--fetch] [--as-common-
↳datasrc NAME] [--publish-depends SIBLINGNAME] [--publish-by-default REFSPEC] [--
↳annex-wanted EXPR] [--annex-required EXPR] [--annex-group EXPR] [--annex-
↳groupwanted EXPR] [--inherit] [--no-annex-info] [-r] [--recursion-limit LEVELS] 
↳[ACTION]
```

### **Description**

Manage sibling configuration

This command offers four different actions: ‘query’, ‘add’, ‘remove’, ‘configure’. ‘query’ is the default action and can be used to obtain information about (all) known siblings. ‘add’ and ‘configure’ are highly similar actions, the only difference being that adding a sibling with a name that is already registered will fail, whereas re-configuring a (different) sibling under a known name will not be considered an error. Lastly, the ‘remove’ action allows for the removal (or de-configuration) of a registered sibling.

For each sibling (added, configured, or queried) all known sibling properties are reported. This includes:

“**name**” Name of the sibling

“**path**” Absolute path of the dataset

“**url**” For regular siblings at minimum a “fetch” URL, possibly also a “pushurl”

Additionally, any further configuration will also be reported using a key that matches that in the Git configuration.

By default, sibling information is rendered as one line per sibling following this scheme:

```
<dataset_path>: <sibling_name>(<+|->) [<access_specification>
```

where the + and - labels indicate the presence or absence of a remote data annex at a particular remote, and ACCESS\_SPECIFICATION contains either a URL and/or a type label for the sibling.

## Options

### **ACTION**

command action selection (see general documentation). Constraints: value must be one of ('query', 'add', 'remove', 'configure') [Default: 'query']

### **-version**

show the program's version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, -dataset DATASET**

specify the dataset to configure. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-s NAME, -name NAME**

name of the sibling. For sibling removal this option is mandatory, otherwise the hostname part of a given URL is used as a default. This option can be used to limit 'query' to a specific sibling. Constraints: value must be a string [Default: None]

### **-url [URL]**

the URL of or path to the dataset sibling named by NAME. For recursive operation it is required that a template string for building subdataset sibling URLs is given. List of currently available placeholders: %NAME the name of the dataset, where slashes are replaced by dashes. Constraints: value must be a string [Default: None]

**–pushurl *PUSHURL***

in case the URL cannot be used to publish to the dataset sibling, this option specifies a URL to be used instead. If no URL is given, PUSHURL serves as URL as well. Constraints: value must be a string [Default: None]

**-D *DESCRIPTION*, –description *DESCRIPTION***

short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string [Default: None]

**–fetch**

fetch the sibling after configuration. [Default: False]

**–as-common-datasrc *NAME***

configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto-enabled special remote). [Default: None]

**–publish-depends *SIBLINGNAME***

add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item ‘remote.SIBLINGNAME .datalad-publish-depends’. This option can be given more than once to configure multiple dependencies. Constraints: value must be a string [Default: None]

**–publish-by-default *REFSPEC***

add a refspec to be published to this sibling by default if nothing specified. Constraints: value must be a string [Default: None]

**–annex-wanted *EXPR***

expression to specify ‘wanted’ content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-wanted/> for more information. Constraints: value must be a string [Default: None]

**–annex-required *EXPR***

expression to specify ‘required’ content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-required/> for more information. Constraints: value must be a string [Default: None]

**–annex-group *EXPR***

expression to specify a group for the repository. See <https://git-annex.branchable.com/git-annex-group/> for more information. Constraints: value must be a string [Default: None]

**–annex-groupwanted EXPR**

expression for the groupwanted. Makes sense only if `–annex-wanted=“groupwanted”` and `annex-group` is given too. See <https://git-annex.branchable.com/git-annex-groupwanted/> for more information. Constraints: value must be a string [Default: None]

**–inherit**

if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: False]

**–no-annex-info**

Whether to query all information about the annex configurations of siblings. Can be disabled if speed is a concern. [Default: True]

**-r, –recursive**

if set, recurse into potential subdataset. [Default: False]

**–recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**Authors**

datalad is developed by The DataLad Team and Contributors <[team@datalad.org](mailto:team@datalad.org)>.

**datalad-subdatasets****Synopsis**

```
datalad-subdatasets [--version] [-h] [-l LEVEL] [--pbs-runner {condor}] [-d DATASET]
↳ [--fulfilled FULFILLED] [-r] [--recursion-limit LEVELS] [--contains PATH] [--
↳ bottomup] [--set-property VALUE VALUE] [--delete-property NAME]
```

**Description**

Report subdatasets and their properties.

The following properties are reported (if possible) for each matching subdataset record.

“**name**” Name of the subdataset in the parent (often identical with the relative path in the parent dataset)

“**path**” Absolute path to the subdataset

“**parentds**” Absolute path to the parent dataset

“**revision**” SHA1 of the subdataset commit recorded in the parent dataset

“**state**” Condition of the subdataset: ‘clean’, ‘modified’, ‘absent’, ‘conflict’ as reported by *git submodule*

“**revision\_descr**” Output of *git describe* for the subdataset

“**gitmodule\_url**” URL of the subdataset recorded in the parent

“**gitmodule\_<label>**” Any additional configuration property on record.

Performance note: Requesting BOTTOMUP reporting order, or a particular numerical RECURSION\_LIMIT implies an internal switch to an alternative query implementation for recursive query that is more flexible, but also notably slower (performs one call to Git per dataset versus a single call for all combined).

## Options

### **-version**

show the program’s version and license information

### **-h, -help, -help-np**

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

### **-l LEVEL, -log-level LEVEL**

set logging verbosity level. Choose among critical, error, warning, info, debug. Also you can specify an integer <10 to provide even more debugging information

### **-pbs-runner {condor}**

execute command by scheduling it via available PBS. For settings, config file will be consulted

### **-d DATASET, -dataset DATASET**

specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) [Default: None]

### **-fulfilled FULFILLED**

if given, must be a boolean flag indicating whether to report either only locally present or absent datasets. By default subdatasets are reported regardless of their status. Constraints: value must be convertible to type bool [Default: None]

### **-r, -recursive**

if set, recurse into potential subdataset. [Default: False]



**–recursion-limit LEVELS**

limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’ [Default: None]

**–contains PATH**

limit report to the subdatasets containing the given path. If a root path of a subdataset is given the last reported dataset will be the subdataset itself. Constraints: value must be a string [Default: None]

**–bottomup**

whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down. [Default: False]

**–set-property VALUE VALUE**

Name and value of one or more subdataset properties to be set in the parent dataset’s .gitmodules file. The value can be a Python format() template string wrapped in ‘<>’ (e.g. ‘<{gitmodule\_name}>’). Supported keywords are any item reported in the result properties of this command, plus ‘refds\_relpath’ and ‘refds\_relname’: the relative path of a subdataset with respect to the base dataset of the command call, and, in the latter case, the same string with all directory separators replaced by dashes. This option can be given multiple times. Constraints: value must be a string [Default: None]

**–delete-property NAME**

Name of one or more subdataset properties to be removed from the parent dataset’s .gitmodules file. This option can be given multiple times. Constraints: value must be a string [Default: None]

**Authors**

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

**Python module reference**

This module reference extends the manual with a comprehensive overview of the available functionality built into datalad. Each module in the package is documented by a general summary of its purpose and the list of classes and functions it provides.

**High-level user interface****Dataset operations**


---

<code>api.add([path, dataset, to_git, save, ...])</code>	Add files/directories to an existing dataset.
----------------------------------------------------------	-----------------------------------------------

Continued on next page

Table 5.1 – continued from previous page

<code>api.create([path, force, description, ...])</code>	Create a new dataset from scratch.
<code>api.create_sibling(sshurl[, name, ...])</code>	Create a dataset sibling on a UNIX-like SSH-accessible machine
<code>api.create_sibling_github(reponame[, ...])</code>	Create dataset sibling on Github.
<code>api.drop([path, dataset, recursive, ...])</code>	Drop file content from datasets
<code>api.export(astype, dataset[, getcmdhelp, output])</code>	Export a dataset to another representation
<code>api.get([path, source, dataset, recursive, ...])</code>	Get any dataset content (files/directories/subdatasets).
<code>api.install([path, source, dataset, ...])</code>	Install a dataset from a (remote) source.
<code>api.publish([path, dataset, to, since, ...])</code>	Publish a dataset to a known <i>sibling</i> .
<code>api.remove([path, dataset, recursive, ...])</code>	Remove components from datasets
<code>api.save([message, files, dataset, ...])</code>	Save the current state of a dataset
<code>api.update([path, sibling, merge, dataset, ...])</code>	Update a dataset from a sibling.
<code>api.uninstall([path, dataset, recursive, ...])</code>	Uninstall subdatasets
<code>api.unlock([path, dataset, recursive, ...])</code>	Unlock file(s) of a dataset

## datalad.api.add

`datalad.api.add` (*path=None, dataset=None, to\_git=None, save=True, message=None, recursive=False, recursion\_limit=None, ds2super=False, git\_opts=None, annex\_opts=None, annex\_add\_opts=None, jobs=None*)

Add files/directories to an existing dataset.

Typically, files and directories to be added to a dataset would be placed into a directory of a dataset, and subsequently this command can be used to register this new content with the dataset. With recursion enabled, files will be added to their respective subdatasets as well.

By default all files are added to the dataset's *annex*, i.e. only their content identity and availability information is tracked with Git. This results in lightweight datasets. If desired, the *to\_git* flag can be used to tell datalad to inject files directly into Git. While this is not recommended for binary data or large files, it can be used for source code and meta-data to be able to benefit from Git's track and merge capabilities. Files checked directly into Git are always and unconditionally available immediately after installation of a dataset.

---

**Note:** Power-user info: This command uses `git annex add`, or `git add` to incorporate new dataset content.

---

### Parameters

- **path** (*non-empty sequence of str or None, optional*) – path/name of the component to be added. The component must either exist on the filesystem already, or a *source* has to be provided. Constraints: value must be a string, or value must be *None*. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the add operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **to\_git** (*bool, optional*) – flag whether to add data directly to Git, instead of tracking data identity only. Usually this is not desired, as it inflates dataset sizes and impacts flexibility of data transport. If not specified - it will be up to git-annex to decide, possibly on `.gitattributes` options. [Default: None]

- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. Constraints: value must be a string, or value must be *None*. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **ds2super** (*bool, optional*) – given paths of dataset (toplevel) locations will cause these datasets to be added to their respective superdatasets underneath a given base *dataset* (instead of all their content to themselves). If no base *dataset* is provided, this flag has no effect. Regular files and directories are always added to their respective datasets, regardless of this setting. [Default: False]
- **git\_opts** (*str or None, optional*) – option string to be passed to **git** calls. Constraints: value must be a string, or value must be *None*. [Default: None]
- **annex\_opts** (*str or None, optional*) – option string to be passed to **git annex** calls. Constraints: value must be a string, or value must be *None*. [Default: None]
- **annex\_add\_opts** (*str or None, optional*) – option string to be passed to **git annex add** calls. Constraints: value must be a string, or value must be *None*. [Default: None]
- **jobs** (*int or None, optional*) – how many parallel jobs (where possible) to use. Constraints: value must be convertible to type ‘int’, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: None]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: None]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of

(‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item- or-list’). [Default: ‘list’]

## datalad.api.create

```
datalad.api.create (path=None, force=False, description=None, dataset=None, no_annex=False,
                  save=True, annex_version=None, annex_backend='MD5E', native_metadata_type=None,
                  shared_access=None, git_opts=None, annex_opts=None, annex_init_opts=None)
```

Create a new dataset from scratch.

This command initializes a new *dataset* at a given location, or the current directory. The new dataset can optionally be registered in an existing *superdataset* (the new dataset’s path needs to be located within the superdataset for that, and the superdataset needs to be given explicitly). It is recommended to provide a brief description to label the dataset’s nature *and* location, e.g. “Michael’s music on black laptop”. This helps humans to identify data locations in distributed scenarios. By default an identifier comprised of user and machine name, plus path will be generated.

This command only creates a new dataset, it does not add any content to it, even if the target directory already contains additional files or directories.

Plain Git repositories can be created via the *no\_annex* flag. However, the result will not be a full dataset, and, consequently, not all features are supported (e.g. a description).

To create a local version of a remote dataset use the *install()* command instead.

---

**Note:** Power-user info: This command uses **git init**, and **git annex init** to prepare the new dataset. Registering to a superdataset is performed via a **git submodule add** operation in the discovered superdataset.

---

### Parameters

- **path** (*str or Dataset or None, optional*) – path where the dataset shall be created, directories will be created as necessary. If no location is provided, a dataset will be created in the current working directory. Either way the command will error if the target directory is not empty. Use *force* to create a dataset in a non-empty directory. Constraints: list expected, each value must be a string, or Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **force** (*bool, optional*) – enforce creation of a dataset in a non-empty directory. [Default: False]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string, or value must be *None*. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the create operation on. If a dataset is give, a new subdataset will be created in it. Constraints:

Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]

- **no\_annex** (*bool, optional*) – if set, a plain Git repository will be created without any annex. [Default: *False*]
- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: *True*]
- **annex\_version** (*int or None, optional*) – select a particular annex repository version. The list of supported versions depends on the available git-annex version. This should be left untouched, unless you know what you are doing. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: *None*]
- **annex\_backend** (*1-item sequence of str or None, optional*) – set default hashing backend used by the new dataset. For a list of supported backends see the git-annex documentation. The default is optimized for maximum compatibility of datasets across platforms (especially those with limited path lengths). Constraints: value must be a string, or value must be *None*. [Default: ‘MD5E’]
- **native\_metadata\_type** (*list of str or None, optional*) – Metadata type label. Must match the name of the respective parser implementation in Datalad (e.g. “bids”). Constraints: list expected, each value must be a string, or value must be *None*. [Default: *None*]
- **shared\_access** – configure shared access to a dataset, see *git init –shared* documentation for complete details on the supported scenarios. Possible values include: ‘false’, ‘true’, ‘group’, and ‘all’. [Default: *None*]
- **git\_opts** (*str or None, optional*) – option string to be passed to **git** calls. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **annex\_opts** (*str or None, optional*) – option string to be passed to **git annex** calls. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **annex\_init\_opts** (*str or None, optional*) – option string to be passed to **git annex init** calls. Constraints: value must be a string, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]

- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'}* 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. Constraints: value must be one of ('paths', 'relpaths', 'datasets', 'successdatasets-or-none'), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item-or-list'). [Default: 'list']

## datalad.api.create\_sibling

```
datalad.api.create_sibling(sshurl, name=None, target_dir=None, target_url=None, target_pushurl=None, dataset=None, recursive=False, recursion_limit=None, existing='error', shared=None, ui=False, as_common_datasrc=None, publish_by_default=None, publish_depends=None, annex_wanted=None, annex_group=None, annex_groupwanted=None, inherit=False, since=None)
```

Create a dataset sibling on a UNIX-like SSH-accessible machine

Given a local dataset, and SSH login information this command creates a remote dataset repository and configures it as a dataset sibling to be used as a publication target (see *publish* command).

Various properties of the remote sibling can be configured (e.g. name location on the server, read and write access URLs, and access permissions).

Optionally, a basic web-viewer for DataLad datasets can be installed at the remote location.

This command supports recursive processing of dataset hierarchies, creating a remote sibling for each dataset in the hierarchy. By default, remote siblings are created in hierarchical structure that reflects the organization on the local file system. However, a simple templating mechanism is provided to produce a flat list of datasets (see *-target-dir*).

### Parameters

- **sshurl** (*str*) – Login information for the target server. This can be given as a URL (*ssh://host/path*) or SSH-style (*user@host:path*). Unless overridden, this also serves the future dataset's access URL and path on the server. Constraints: list expected, each value must be a string.
- **name** (*str* or *None*, *optional*) – sibling name to create for this publication target. If *recursive* is set, the same name will be used to label all the subdatasets' siblings. When creating a target dataset fails, no sibling is added. Constraints: list expected, each value must be a string, or value must be *None*. [Default: *None*]
- **target\_dir** (*str* or *None*, *optional*) – path to the directory *on the server* where the dataset shall be created. By default the SSH access URL is used to identify this directory. If a relative path is provided here, it is interpreted as being relative to the user's home directory on the server. Additional features are relevant for recursive processing of datasets with subdatasets. By default, the local dataset structure is replicated on the server. However, it is possible to provide a template for generating different target directory names for all (sub)datasets. Templates can contain certain placeholder that are substituted for each

(sub)dataset. For example: “/mydirectory/dataset%%RELNAME”. Supported placeholders: %%RELNAME - the name of the datasets, with any slashes replaced by dashes . Constraints: value must be a string, or value must be *None*. [Default: None]

- **target\_url** (*str* or *None*, *optional*) – “public” access URL of the to-be-created target dataset(s) (default: *sshurl*). Accessibility of this URL determines the access permissions of potential consumers of the dataset. As with *target\_dir*, templates (same set of placeholders) are supported. Also, if specified, it is provided as the annex description . Constraints: value must be a string, or value must be *None*. [Default: None]
- **target\_pushurl** (*str* or *None*, *optional*) – In case the *target\_url* cannot be used to publish to the dataset, this option specifies an alternative URL for this purpose. As with *target\_url*, templates (same set of placeholders) are supported. . Constraints: value must be a string, or value must be *None*. [Default: None]
- **dataset** (*Dataset* or *None*, *optional*) – specify the dataset to create the publication target for. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **recursive** (*bool*, *optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int* or *None*, *optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **existing** (*{'skip', 'replace', 'error', 'reconfigure'}*, *optional*) – action to perform, if a sibling is already configured under the given name and/or a target directory already exists. In this case, a dataset can be skipped (‘skip’), an existing target directory be forcefully re-initialized, and the sibling (re-)configured (‘replace’, implies ‘reconfigure’), the sibling configuration be updated only (‘reconfigure’), or to error (‘error’). Constraints: value must be one of (‘skip’, ‘replace’, ‘error’, ‘reconfigure’). [Default: ‘error’]
- **shared** (*str* or *bool* or *None*, *optional*) – if given, configures the access permissions on the server for multi- users (this could include access by a webserver!). Possible values for this option are identical to those of *git init -shared* and are described in its documentation. Constraints: value must be a string, or value must be convertible to type bool, or value must be *None*. [Default: None]
- **ui** (*bool* or *str*, *optional*) – publish a web interface for the dataset with an optional user- specified name for the html at publication target. defaults to *index.html* at dataset root. Constraints: value must be convertible to type bool, or value must be a string. [Default: False]
- **as\_common\_datasrc** – configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto-enabled special remote). [Default: None]
- **publish\_by\_default** (*list of str* or *None*, *optional*) – add a refspec to be published to this sibling by default if nothing specified. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **publish\_depends** (*list of str* or *None*, *optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item ‘remote.SIBLINGNAME.datalad-publish-depends’. Multiple dependencies can be given as a list of sibling names. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]

- **annex\_wanted** (*str* or *None*, *optional*) – expression to specify ‘wanted’ content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-wanted/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **annex\_group** (*str* or *None*, *optional*) – expression to specify a group for the repository. See <https://git-annex.branchable.com/git-annex-group/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **annex\_groupwanted** (*str* or *None*, *optional*) – expression for the group-wanted. Makes sense only if `annex_wanted="groupwanted"` and `annex-group` is given too. See <https://git-annex.branchable.com/git-annex-groupwanted/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **inherit** (*bool*, *optional*) – if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: *False*]
- **since** (*str* or *None*, *optional*) – limit processing to datasets that have been changed since a given state (by tag, branch, commit, etc). This can be used to create siblings for recently added subdatasets. Constraints: value must be a string, or value must be *None*. [Default: *None*]

### datalad.api.create\_sibling\_github

```
datalad.api.create_sibling_github(reponame, dataset=None, recursive=False, recursion_limit=None, name='github', existing='error', github_login=None, github_passwd=None, github_organization=None, access_protocol='https', publish_depends=None, dryrun=False)
```

Create dataset sibling on Github.

A repository can be created under a user’s Github account, or any organization a user is a member of (given appropriate permissions).

Recursive sibling creation for subdatasets is supported. A dataset hierarchy is represented as a flat list of Github repositories.

Github cannot host dataset content. However, in combination with other data sources (and siblings), publishing a dataset to Github can facilitate distribution and exchange, while still allowing any dataset consumer to obtain actual data content from alternative sources.

For Github authentication user credentials can be given as arguments. Alternatively, they are obtained interactively or queried from the systems credential store. Lastly, an *oauth* token stored in the Git configuration under variable *hub.oauthtoken* will be used automatically. Such a token can be obtained, for example, using the commandline Github interface (<https://github.com/sociomantic/git-hub>) by running: `git hub setup`.

#### Parameters

- **reponame** (*str*) – Github repository name. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string.
- **dataset** (*Dataset* or *None*, *optional*) – specify the dataset to create the publication target for. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a *Dataset* or a valid identifier of a *Dataset* (e.g. a path), or value must be *None*. [Default: *None*]
- **recursive** (*bool*, *optional*) – if set, recurse into potential subdataset. [Default: *False*]



- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **name** (*str, optional*) – name to represent the Github repository in the local dataset installation. Constraints: value must be a string. [Default: ‘github’]
- **existing** (*{‘skip’, ‘error’, ‘reconfigure’}, optional*) – desired behavior when already existing or configured siblings are discovered. ‘skip’: ignore; ‘error’: fail immediately; ‘reconfigure’: use the existing repository and reconfigure the local dataset to use it as a sibling. Constraints: value must be one of (‘skip’, ‘error’, ‘reconfigure’). [Default: ‘error’]
- **github\_login** (*str or None, optional*) – Github user name or access token. Constraints: value must be a string, or value must be *None*. [Default: None]
- **github\_passwd** (*str or None, optional*) – Github user password. Constraints: value must be a string, or value must be *None*. [Default: None]
- **github\_organization** (*str or None, optional*) – If provided, the repository will be created under this Github organization. The respective Github user needs appropriate permissions. Constraints: value must be a string, or value must be *None*. [Default: None]
- **access\_protocol** (*{‘https’, ‘ssh’}, optional*) – Which access protocol/URL to configure for the sibling. Constraints: value must be one of (‘https’, ‘ssh’). [Default: ‘https’]
- **publish\_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item ‘remote.SIBLINGNAME.datalad-publish-depends’. Multiple dependencies can be given as a list of sibling names. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **dryrun** (*bool, optional*) – If this flag is set, no communication with Github is performed, and no repositories will be created. Instead would-be repository names are reported for all relevant datasets . [Default: False]

## datalad.api.drop

`datalad.api.drop` (*path=None, dataset=None, recursive=False, recursion\_limit=None, check=True, if\_dirty='save-before'*)

Drop file content from datasets

This command takes any number of paths of files and/or directories. If a common (super)dataset is given explicitly, the given paths are interpreted relative to this dataset.

Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

By default, the availability of at least one remote copy is verified, before file content is dropped. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

## Examples

Drop all file content in a dataset:

```
~/some/dataset$ datalad drop
```

Drop all file content in a dataset and all its subdatasets:

```
~/some/dataset$ datalad drop --recursive
```

### Parameters

- **path** (*sequence of str or None, optional*) – path/name of the component to be dropped. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: *None*]
- **check** (*bool, optional*) – whether to perform checks to assure the configured minimum number (remote) source for data. [Default: *True*]
- **if\_dirty** – desired behavior if a dataset with unsaved changes is discovered: ‘fail’ will trigger an error and further processing is aborted; ‘save-before’ will save all changes prior any further action; ‘ignore’ let’s datalad proceed as if the dataset would not have unsaved changes. [Default: ‘save-before’]
- **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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item- or-list’). [Default: ‘list’]

## datalad.api.export

`datalad.api.export` (*astype, dataset, getcmdhelp=False, output=None, \*\*kwargs*)  
Export a dataset to another representation

### Parameters

- **astype** – label of the type or format the dataset shall be exported to.
- **dataset** (*Dataset or None*) – specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*.
- **getcmdhelp** (*bool, optional*) – show help for a specific export type/format. [Default: False]
- **output** – output destination specification to be passes to the exporter. The particular semantics of the option value depend on the actual exporter. Typically, this will be a file name or a path to a directory. [Default: None]

## datalad.api.get

`datalad.api.get` (*path=None, source=None, dataset=None, recursive=False, recursion\_limit=None, get\_data=True, description=None, reckless=False, jobs=None, verbose=False*)  
Get any dataset content (files/directories/subdatasets).

This command only operates on dataset content. To obtain a new independent dataset from some source use the *install* command.

By default this command operates recursively within a dataset, but not across potential subdatasets, i.e. if a directory is provided, all files in the directory are obtained. Recursion into subdatasets is supported too. If enabled, relevant subdatasets are detected and installed in order to fulfill a request.

Known data locations for each requested file are evaluated and data are obtained from some available location (according to git-annex configuration and possibly assigned remote priorities), unless a specific source is specified.

---

**Note:** Power-user info: This command uses `git annex get` to fulfill file handles.

---

### Parameters

- **path** (*sequence of str or None, optional*) – path/name of the requested dataset component. The component must already be known to a dataset. To add new components to a dataset use the *add* command. Constraints: value must be a string, or value must be *None*. [Default: None]
- **source** (*str or None, optional*) – label of the data source to be used to fulfill requests. This can be the name of a dataset *sibling* or another known source. Constraints: value must be a string, or value must be *None*. [Default: None]

- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the add operation on, in which case *path* arguments are interpreted as being relative to this dataset. If no dataset is given, an attempt is made to identify a dataset for each input *path*. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or {'existing'} or None, optional*) – limit recursion into subdataset to the given number of levels. Alternatively, ‘existing’ will limit recursion to subdatasets that already existed on the filesystem at the start of processing, and prevent new subdatasets from being obtained recursively. Constraints: value must be convertible to type ‘int’, or value must be one of (‘existing’), or value must be *None*. [Default: None]
- **get\_data** (*bool, optional*) – whether to obtain data for all file handles. If disabled, *get* operations are limited to dataset handles. [Default: True]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string, or value must be *None*. [Default: None]
- **reckless** (*bool, optional*) – Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: False]
- **jobs** (*int or None, optional*) – how many parallel jobs (where possible) to use. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **verbose** (*bool, optional*) – print out more detailed information while executing a command. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: None]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: None]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of ('paths', 'relpaths', 'datasets', 'successdatasets-or-none'), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item-or-list'). [Default: 'list']

## datalad.api.install

```
datalad.api.install (path=None, source=None, dataset=None, get_data=False, description=None, recursive=False, recursion_limit=None, save=True, reckless=False, jobs=None)
```

Install a dataset from a (remote) source.

This command creates a local *sibling* of an existing dataset from a (remote) location identified via a URL or path. Optional recursion into potential subdatasets, and download of all referenced data is supported. The new dataset can be optionally registered in an existing *superdataset* by identifying it via the *dataset* argument (the new dataset's path needs to be located within the superdataset for that).

It is recommended to provide a brief description to label the dataset's nature *and* location, e.g. "Michael's music on black laptop". This helps humans to identify data locations in distributed scenarios. By default an identifier comprised of user and machine name, plus path will be generated.

When only partial dataset content shall be obtained, it is recommended to use this command without the *get-data* flag, followed by a *get ()* operation to obtain the desired data.

---

**Note:** Power-user info: This command uses **git clone**, and **git annex init** to prepare the dataset. Registering to a superdataset is performed via a **git submodule add** operation in the discovered superdataset.

---

### Parameters

- **path** – path/name of the installation target. If no *path* is provided a destination path will be derived from a source URL similar to **git clone**. [Default: *None*]
- **source** (*str or None, optional*) – URL or local path of the installation source. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the install operation on. If no dataset is given, an attempt is made to identify the dataset in a parent directory of the current working directory and/or the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **get\_data** (*bool, optional*) – if given, obtain all data content too. [Default: *False*]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string, or value must be *None*. [Default: *None*]

- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]
- **reckless** (*bool, optional*) – Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: False]
- **jobs** (*int or None, optional*) – how many parallel jobs (where possible) to use. Constraints: value must be convertible to type ‘int’, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: None]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: None]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item-or-list’). [Default: ‘list’]

## datalad.api.publish

`datalad.api.publish` (*path=None, dataset=None, to=None, since=None, missing='fail', force=False, recursive=False, recursion\_limit=None, git\_opts=None, annex\_opts=None, annex\_copy\_opts=None*)

Publish a dataset to a known *sibling*.

This makes the last saved state of a dataset available to a sibling or special remote data store of a dataset. Any target sibling must already exist and be known to the dataset.

Optionally, it is possible to limit publication to change sets relative to a particular point in the version history of a dataset (e.g. a release tag). By default, the state of the local dataset is evaluated against the last known state of the target sibling. Actual publication is only attempted if there was a change compared to the reference state, in order to speed up processing of large collections of datasets. Evaluation with respect to a particular “historic” state is only supported in conjunction with a specified reference dataset. Change sets are also evaluated recursively, i.e. only those subdatasets are published where a change was recorded that is reflected in to current state of the top-level reference dataset. See “since” option for more information.

Only publication of saved changes is supported. Any unsaved changes in a dataset (hierarchy) have to be saved before publication.

---

**Note:** Power-user info: This command uses `git push`, and `git annex copy` to publish a dataset. Publication targets are either configured remote Git repositories, or git-annex special remotes (if their support data upload).

---

### Parameters

- **path** (*sequence of str or None, optional*) – path(s), that may point to file handle(s) to publish including their actual content or to subdataset(s) to be published. If a file handle is published with its data, this implicitly means to also publish the (sub)dataset it belongs to. ‘.’ as a path is treated in a special way in the sense, that it is passed to subdatasets in case *recursive* is also given. Constraints: value must be a string, or value must be *None*. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the (top-level) dataset to be published. If no dataset is given, the datasets are determined based on the input arguments. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **to** (*str or None, optional*) – name of the target sibling. If no name is given an attempt is made to identify the target based on the dataset’s configuration (i.e. a configured tracking branch, or a single sibling that is configured for publication). Constraints: value must be a string, or value must be *None*. [Default: None]
- **since** (*str or None, optional*) – When publishing dataset(s), specifies commit (treeish, tag, etc) from which to look for changes to decide either updated publishing is necessary for this and which children. If empty argument is provided, then we will always run publish command. By default, would take from the previously published to that remote/sibling state (for the current branch). Constraints: value must be a string, or value must be *None*. [Default: None]
- **missing** (*{'fail', 'inherit', 'skip'}, optional*) – action to perform, if a sibling does not exist in a given dataset. By default it would fail the run (‘fail’ setting). With ‘inherit’ a ‘create-sibling’ with ‘–inherit-settings’ will be used to create sibling on the remote. With ‘skip’ - it simply will be skipped. Constraints: value must be one of (‘fail’, ‘inherit’, ‘skip’). [Default: ‘fail’]

- **force** (*bool, optional*) – enforce doing publish activities (git push etc) regardless of the analysis if they seemed needed. [Default: False]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: None]
- **git\_opts** (*str or None, optional*) – option string to be passed to **git** calls. Constraints: value must be a string, or value must be *None*. [Default: None]
- **annex\_opts** (*str or None, optional*) – option string to be passed to **git annex** calls. Constraints: value must be a string, or value must be *None*. [Default: None]
- **annex\_copy\_opts** (*str or None, optional*) – option string to be passed to **git annex copy** calls. Constraints: value must be a string, or value must be *None*. [Default: None]

## datalad.api.remove

`datalad.api.remove` (*path=None, dataset=None, recursive=False, check=True, if\_dirty='save-before'*)  
 Remove components from datasets

This command can remove any components (subdatasets, and (directories with) files) from datasets. Removing a component implies any present content to be dropped, and any associated subdatasets to be uninstalled. Subsequently, the component is “unregistered” from the respective dataset. This means that the respective component is no longer present on the file system.

By default, the availability of at least one remote copy is verified, by default, before file content is dropped. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

Any number of paths to process can be given as input. Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

## Examples

Permanently remove a subdataset from a dataset and wipe out the subdataset association too:

```
~/some/dataset$ datalad remove somesubdataset1
```

## Parameters

- **path** (*sequence of str or None, optional*) – path/name of the component to be removed. Constraints: value must be a string, or value must be *None*. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **check** (*bool, optional*) – whether to perform checks to assure the configured minimum number (remote) source for data. [Default: True]



- **if\_dirty** – desired behavior if a dataset with unsaved changes is discovered: ‘fail’ will trigger an error and further processing is aborted; ‘save-before’ will save all changes prior any further action; ‘ignore’ let’s datalad proceed as if the dataset would not have unsaved changes. [Default: ‘save-before’]
- **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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be `None`. [Default: `None`]
- **result\_renderer** (*{‘default’, ‘json’, ‘json\_pp’, ‘tailored’} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be `None`. [Default: `None`]
- **result\_xfm** (*{‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be `None`. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item- or-list’). [Default: ‘list’]

## datalad.api.save

`datalad.api.save` (*message=None, files=None, dataset=None, all\_updated=True, all\_changes=None, version\_tag=None, recursive=False, recursion\_limit=None, super\_datasets=False*)  
Save the current state of a dataset

Saving the state of a dataset records all changes that have been made to it. This change record is annotated with a user-provided description. Optionally, an additional tag, such as a version, can be assigned to the saved state. Such tag enables straightforward retrieval of past versions at a later point in time.

**Returns** `None` if nothing was saved, the resulting commit otherwise.

**Return type** `commit` or `None`

### Parameters

- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. Constraints: value must be a string, or value must be `None`. [Default: `None`]

- **files** (*sequence of str or None, optional*) – list of files to consider. If given, only changes made to those files are recorded in the new state. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – “specify the dataset to save. If a dataset is given, but no *files*, the entire dataset will be saved. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **all\_updated** (*bool, optional*) – if no explicit paths are given, save changes of all known components in a datasets. [Default: *True*]
- **all\_changes** (*bool, optional*) – save all changes (even to not yet added files) of all components in datasets that contain any of the given paths [DEPRECATED!]. [Default: *None*]
- **version\_tag** (*str or None, optional*) – an additional marker for that state. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: *None*]
- **super\_datasets** (*bool, optional*) – if set, save a change in a dataset also in its superdataset. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item- or-list'). [Default: 'list']

## datalad.api.update

`datalad.api.update` (*path=None, sibling=None, merge=False, dataset=None, recursive=False, recursion\_limit=None, fetch\_all=False, reobtain\_data=False*)

Update a dataset from a sibling.

### Parameters

- **path** (*sequence of str or None, optional*) – path to be updated. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **sibling** (*str or None, optional*) – name of the sibling to update from. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **merge** (*bool, optional*) – merge obtained changes from the given or the default sibling. [Default: *False*]
- **dataset** (*Dataset or None, optional*) – “specify the dataset to update. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: *None*]
- **fetch\_all** (*bool, optional*) – fetch updates from all known siblings. [Default: *False*]
- **reobtain\_data** (*bool, optional*) – TODO. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of ('paths', 'relpaths', 'datasets', 'successdatasets-or-none'), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item-or-list'). [Default: 'list']

## datalad.api.uninstall

`datalad.api.uninstall` (*path=None, dataset=None, recursive=False, check=True, if\_dirty='save-before'*)

Uninstall subdatasets

This command can be used to uninstall any number of installed subdataset. If a to-be-uninstalled subdataset contains presently installed subdatasets itself, their recursive removal has to be enabled explicitly to avoid the command to exit with an error. This command will error if individual files or non-dataset directories are given as input (use the drop or remove command depending in the desired goal), nor will it uninstall top-level datasets (i.e. datasets that or not a subdataset in another dataset; use the remove command for this purpose).

By default, the availability of at least one remote copy for each currently available file in any dataset is verified. As these checks could lead to slow operation (network latencies, etc), they can be disabled.

Any number of paths to process can be given as input. Recursion into subdatasets needs to be explicitly enabled, while recursion in subdirectories within a dataset as always done automatically. An optional recursion limit is applied relative to each given input path.

## Examples

Uninstall a subdataset (undo installation):

```
~/some/dataset$ datalad uninstall somesubdataset1
```

## Parameters

- **path** (*sequence of str or None, optional*) – path/name of the component to be uninstalled. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the operation on. If no dataset is given, an attempt is made to identify a dataset based on the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **check** (*bool, optional*) – whether to perform checks to assure the configured minimum number (remote) source for data. [Default: *True*]
- **if\_dirty** – desired behavior if a dataset with unsaved changes is discovered: 'fail' will trigger an error and further processing is aborted; 'save-before' will save all changes prior any further action; 'ignore' let's datalad proceed as if the dataset would not have unsaved changes. [Default: 'save-before']

- **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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be `None`. [Default: `None`]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be `None`. [Default: `None`]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be `None`. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item-or-list’). [Default: ‘list’]

## datalad.api.unlock

`datalad.api.unlock` (*path=None, dataset=None, recursive=False, recursion\_limit=None*)

Unlock file(s) of a dataset

Unlock files of a dataset in order to be able to edit the actual content

### Parameters

- **path** (*sequence of str or None, optional*) – file(s) to unlock. Constraints: value must be a string, or value must be `None`. [Default: `None`]
- **dataset** (*Dataset or None, optional*) – “specify the dataset to unlock files in. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. If the latter fails, an attempt is made to identify the dataset based on `path`. Constraints: Value must be a `Dataset` or a valid identifier of a `Dataset` (e.g. a path), or value must be `None`. [Default: `None`]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: `False`]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value

must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item-or-list’). [Default: ‘list’]

## Meta data handling

<code>api.search(match[, dataset, search, report, ...])</code>	Search within available in datasets’ meta data
<code>api.aggregate_metadata(dataset[, ...])</code>	Aggregate meta data of a dataset for later query.

## datalad.api.search

`datalad.api.search` (*match, dataset=None, search=None, report=None, report\_matched=False, format='custom', regex=False*)

Search within available in datasets’ meta data

### Yields

- **location** (*str*) – (relative) path to the dataset
- **report** (*dict*) – fields which were requested by *report* option

### Parameters

- **match** – a string (or a regular expression if *regex=True*) to search for in all meta data values. If multiple provided, all must have a match among some fields of a dataset.
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the query operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **search** – name of the property to search for any match. By default, all properties are searched. [Default: None]
- **report** – name of the property to report for any match. If '\*' is given, all properties are reported. [Default: None]
- **report\_matched** (*bool, optional*) – flag to report those fields which have matches. If *report* option values are provided, union of matched and those in *report* will be output. [Default: False]
- **format** (*{'custom', 'json', 'yaml'}, optional*) – format for output. Constraints: value must be one of ('custom', 'json', 'yaml'). [Default: 'custom']
- **regex** (*bool, optional*) – flag for STRING to be used as a (Python) regular expression which should match the value. [Default: False]

### datalad.api.aggregate\_metadata

`datalad.api.aggregate_metadata(dataset, guess_native_type=False, recursive=False, recursion_limit=None, save=True, if_dirty='save-before')`

Aggregate meta data of a dataset for later query.

By default meta data is aggregated across all configured native meta data sources. Optionally, the type of available meta data can be guessed, if no types are configured. Moreover, it is possible to aggregate meta data from any subdatasets into the superdataset, in order to facilitate data discovery without having to obtain any subdataset.

**Returns** Any datasets where (updated) aggregated meta data was saved.

**Return type** List

#### Parameters

- **dataset** (*Dataset or None*) – specify the dataset to perform the install operation on. If no dataset is given, an attempt is made to identify the dataset based on the current working directory and/or the *path* given. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*.
- **guess\_native\_type** (*bool, optional*) – guess native meta data type of datasets, if none is configured. With a configured, or auto-detected meta data type, no native meta data will be aggregated. [Default: False]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int', or value must be *None*. [Default: None]
- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Given this option will disable this behavior. [Default: True]

- **if\_dirty** – desired behavior if a dataset with unsaved changes is discovered: ‘fail’ will trigger an error and further processing is aborted; ‘save-before’ will save all changes prior any further action; ‘ignore’ let’s datalad proceed as if the dataset would not have unsaved changes. [Default: ‘save-before’]

## Plumbing commands

<code>api.annotate_paths([path, dataset, ...])</code>	Analyze and act upon input paths
<code>api.clean([dataset, what, recursive, ...])</code>	Clean up after DataLad (possible temporary files etc.)
<code>api.clone(source[, path, dataset, ...])</code>	Obtain a dataset copy from a URL or local source (path)
<code>api.create_test_dataset([path, spec, seed])</code>	Create test (meta-)dataset.
<code>api.diff([path, dataset, revision, staged, ...])</code>	Report changes of dataset component between revisions.
<code>api.download_url(urls[, path, overwrite, ...])</code>	Download content
<code>api.ls(loc[, recursive, fast, all_, long_, ...])</code>	List summary information about URLs and dataset(s)
<code>api.sshrun(login, cmd[, port, no_stdin])</code>	Run command on remote machines via SSH.
<code>api.siblings([action, dataset, name, url, ...])</code>	Manage sibling configuration
<code>api.subdatasets([dataset, fulfilled, ...])</code>	Report subdatasets and their properties.

## datalad.api.annotate\_paths

```
datalad.api.annotate_paths (path=None, dataset=None, recursive=False, recur-
                             sion_limit=None, action=None, unavailable_path_status='',
                             unavailable_path_msg=None, nondataset_path_status='error',
                             force_parents_discovery=True, force_subds_discovery=True, modi-
                             fied=None)
```

Analyze and act upon input paths

Given paths (or more generally location requests) are inspected and annotated with a number of properties. A list of recognized properties is provided below.

Input *paths* for this command can either be un-annotated (raw) path strings, or already (partially) annotated paths. In the latter case, further annotation is limited to yet-unknown properties, and is potentially faster than initial annotation.

### Recognized path properties

“**action**” label of the action that triggered the path annotation

“**annexkey**” annex key for the content of a file

“**logger**” logger for reporting a message

“**message**” message (plus possible tsring expansion arguments)

“**orig\_request**” original input by which a path was determined

“**parents**” path of dataset containing the annotated path (superdataset for subdatasets)

“**path**” absolute path that is annotated

“**process\_content**” flag that content underneath the path is to be processed

“**process\_updated\_only**” flag that only known dataset components are to be processed

“**raw\_input**” flag whether this path was given as raw (non-annotated) input

“**refds**” path of a reference/base dataset the annotated path is part of

“**registered\_subds**” flag whether a dataset is known to be a true subdataset of *parents*



“**revision**” the recorded commit for a subdataset in a superdataset

“**revision\_descr**” a human-readable description of *revision*

“**source\_url**” URL a dataset was installed from

“**staged**” flag whether a path is known to be “staged” in its containing dataset

“**state**” state indicator for a path in its containing dataset (clean, modified, absent (also for files), conflict)

“**status**” action result status (ok, notneeded, impossible, error)

“**type**” nature of the path (file, directory, dataset)

“**url**” registered URL for a subdataset in a superdataset

In the case of enabled modification detection the results may contain additional properties regarding the nature of the modification. See the documentation of the *diff* command for details.

### Parameters

- **path** (*sequence of str or None, optional*) – path to be annotated. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – an optional reference/base dataset for the paths. Constraints: Value must be a *Dataset* or a valid identifier of a *Dataset* (e.g. a path), or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [Default: *None*]
- **action** (*str or None, optional*) – an “action” property value to include in the path annotation. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **unavailable\_path\_status** (*str or None, optional*) – a “status” property value to include in the annotation for paths that are underneath a dataset, but do not exist on the filesystem. Constraints: value must be a string, or value must be *None*. [Default: ‘’]
- **unavailable\_path\_msg** (*str or None, optional*) – a “message” property value to include in the annotation for paths that are underneath a dataset, but do not exist on the filesystem. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **nondataset\_path\_status** (*str or None, optional*) – a “status” property value to include in the annotation for paths that are not underneath any dataset. Constraints: value must be a string, or value must be *None*. [Default: ‘error’]
- **force\_parents\_discovery** (*bool, optional*) – Flag to disable reports of parent dataset information for any path, in particular dataset root paths. Disabling saves on command run time, if this information is not needed. [Default: *True*]
- **force\_subds\_discovery** (*bool, optional*) – Flag to disable reporting type=‘dataset’ for subdatasets, even when they are not installed, or their mount point directory doesn’t exist. Disabling saves on command run time, if this information is not needed. [Default: *True*]
- **modified** (*str or bool or None, optional*) – comparison reference specification for modification detection. This can be (mostly) anything that *git diff* understands

(commit, treeish, tag, etc). See the documentation of *datalad diff-revision* for details. Unmodified paths will not be annotated. If a requested path was not modified but some content underneath it was, then the request is replaced by the modified paths and those are annotated instead. This option can be used without an argument to test against changes that have been made, but have not yet been staged for a commit. Constraints: list expected, each value must be a string, or value must be convertible to type bool, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item-or-list’). [Default: ‘list’]

## datalad.api.clean

`datalad.api.clean` (*dataset=None, what=None, recursive=False, recursion\_limit=None*)

Clean up after DataLad (possible temporary files etc.)

Removes extracted temporary archives, etc.

### Examples

```
$ datalad clean
```

#### Parameters

- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the clean operation on. If no dataset is given, an attempt is made to identify the dataset in current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **what** – What to clean. If none specified – all known targets are cleaned. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item- or-list’). [Default: ‘list’]

## datalad.api.clone

`datalad.api.clone` (*source, path=None, dataset=None, description=None, reckless=False, alt\_sources=None*)

Obtain a dataset copy from a URL or local source (path)

The purpose of this command is to obtain a new clone (copy) of a dataset and place it into a not-yet-existing or empty directory. As such *clone* provides a strict subset of the functionality offered by *install*. Only a single

dataset can be obtained, recursion is not supported. However, once installed, arbitrary dataset components can be obtained via a subsequent *get* command.

Primary differences over a direct *git clone* call are 1) the automatic initialization of a dataset annex (pure Git repositories are equally supported); 2) automatic registration of the newly obtained dataset as a subdataset (submodule), if a parent dataset is specified; 3) support for datalad’s resource identifiers and automatic generation of alternative access URL for common cases (such as appending ‘.git’ to the URL in case the accessing the base URL failed); and 4) ability to take additional alternative source locations as an argument.

### Parameters

- **source** (*str* or *None*) – URL, Datalad resource identifier, local path or instance of dataset to be cloned. Constraints: value must be a string, or value must be *None*.
- **path** – path to clone into. If no *path* is provided a destination path will be derived from a source URL similar to **git clone**. [Default: *None*]
- **dataset** (*Dataset* or *None, optional*) – (parent) dataset to clone into. If given, the newly cloned dataset is registered as a subdataset of the parent. Also, if given, relative paths are interpreted as being relative to the parent dataset, and not relative to the working directory. Constraints: Value must be a *Dataset* or a valid identifier of a *Dataset* (e.g. a path), or value must be *None*. [Default: *None*]
- **description** (*str* or *None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., “mike’s dataset on lab server”). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **reckless** (*bool, optional*) – Set up the dataset to be able to obtain content in the cheapest/fastest possible way, even if this poses a potential risk the data integrity (e.g. hardlink files from a local clone of the dataset). Use with care, and limit to “read-only” use cases. With this flag the installed dataset will be marked as untrusted. [Default: *False*]
- **alt\_sources** (*non-empty sequence of str* or *None, optional*) – Alternative sources to be tried if a dataset cannot be obtained from the main *source*. Constraints: value must be a string, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'}* or *None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'}* 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. Constraints: value must be one of ('paths', 'relpaths', 'datasets', 'successdatasets-or-none'), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item-or-list'). [Default: 'list']

## datalad.api.create\_test\_dataset

`datalad.api.create_test_dataset` (*path=None, spec=None, seed=None*)

Create test (meta-)dataset.

### Parameters

- **path** (*str or None, optional*) – path/name where to create (if specified, must not exist). Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **spec** (*str or None, optional*) – spec for hierarchy, defined as a min-max (min could be omitted to assume 0) defining how many (random number from min to max) of sub-datasets to generate at any given level of the hierarchy. Each level separated from each other with /. Example: 1-3/-2 would generate from 1 to 3 subdatasets at the top level, and up to two within those at the 2nd level. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **seed** (*int or None, optional*) – seed for rng. Constraints: value must be convertible to type 'int', or value must be *None*. [Default: *None*]

## datalad.api.diff

`datalad.api.diff` (*path=None, dataset=None, revision='HEAD', staged=False, ignore\_subdatasets='none', recursive=False, recursion\_limit=None*)

Report changes of dataset component between revisions.

### Parameters

- **path** (*sequence of str or None, optional*) – path to be evaluated. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **dataset** (*Dataset or None, optional*) – specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **revision** – comparison reference specification. Three modes are supported: 1) <revision> changes you have in your working tree relative to the named revision (this can also be a branch name, tag, commit or any label Git can understand). 2) <revision>..<revision> changes between two arbitrary revisions. 3) <revision>...<revision> changes on the branch containing and up to the second <revision>, starting at a common ancestor of both revisions. [Default: 'HEAD']

- **staged** (*bool, optional*) – get the changes already staged for a commit relative to an optionally given revision (by default the most recent one). [Default: False]
- **ignore\_subdatasets** (*{'none', 'untracked', 'dirty', 'all'}, optional*) – speed up execution by (partially) not evaluating the state of subdatasets in a parent dataset. With “none” a subdataset is considered modified when it either contains untracked or modified content or its last saved state differs from that recorded in the parent dataset. When “untracked” is used subdatasets are not considered modified when they only contain untracked content (but they are still scanned for modified content). Using “dirty” ignores all changes to the work tree of subdatasets, only changes to the revisions stored in the parent dataset are shown. Using “all” hides all changes to subdatasets. Note, even with “all” recursive execution will still report other changes in any existing subdataset, only the subdataset record in a parent dataset is not evaluated. Constraints: value must be one of ('none', 'untracked', 'dirty', 'all'). [Default: 'none']
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: False]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type 'int', or value must be *None*. [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. Constraints: value must be one of ('ignore', 'continue', 'stop'). [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. Constraints: value must be a callable, or value must be *None*. [Default: None]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of ('default', 'json', 'json\_pp', 'tailored'), or value must be *None*. [Default: None]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of ('paths', 'relpaths', 'datasets', 'successdatasets-or-none'), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of ('generator', 'list', 'item-or-list'). [Default: 'list']

## datalad.api.download\_url

`datalad.api.download_url(urls, path=None, overwrite=False, stop_on_failure=False)`

Download content

It allows for a uniform download interface to various supported URL schemes, re-using or asking for authentication detail maintained by datalad.

### Examples

```
$ datalad download http://example.com/file.dat s3://bucket/file2.dat
```

#### Parameters

- **urls** (*non-empty sequence of str*) – URL(s) to be downloaded. Constraints: value must be a string.
- **path** (*str or None, optional*) – path (filename or directory path) where to store downloaded file(s). In case of multiple URLs provided, must point to a directory. Otherwise current directory is used. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **overwrite** (*bool, optional*) – flag to overwrite it if target file exists. [Default: *False*]
- **stop\_on\_failure** (*bool, optional*) – flag to stop subsequent downloads upon first failure to download. [Default: *False*]

**Returns** downloaded successfully files

**Return type** list of str

## datalad.api.ls

`datalad.api.ls(loc, recursive=False, fast=False, all_=False, long_=False, config_file=None, list_content=False, json=None)`

List summary information about URLs and dataset(s)

ATM only s3:// URLs and datasets are supported

### Examples

```
$ datalad ls s3://openfmri/tarballs/ds202 # to list S3 bucket $ datalad ls # to list current dataset
```

#### Parameters

- **loc** (*sequence of str or None*) – URL or path to list, e.g. s3://... Constraints: value must be a string, or value must be *None*.
- **recursive** (*bool, optional*) – recurse into subdirectories. [Default: *False*]
- **fast** (*bool, optional*) – only perform fast operations. Would be overridden by `-all`. [Default: *False*]
- **all** (*bool, optional*) – list all (versions of) entries, not e.g. only latest entries in case of S3. [Default: *False*]

- **long** (*bool, optional*) – list more information on entries (e.g. acl, urls in s3, annex sizes etc). [Default: False]
- **config\_file** (*str or None, optional*) – path to config file which could help the ‘ls’. E.g. for s3:// URLs could be some ~/.s3cfg file which would provide credentials. Constraints: value must be a string, or value must be *None*. [Default: None]
- **list\_content** – list also the content or only first 10 bytes (first10), or md5 checksum of an entry. Might require expensive transfer and dump binary output to your screen. Do not enable unless you know what you are after. [Default: False]
- **json** – metadata json of dataset for creating web user interface. display: prints jsons to stdout or file: writes each subdir metadata to json file in subdir of dataset or delete: deletes all metadata json files in dataset. [Default: None]

## datalad.api.sshrun

`datalad.api.sshrun` (*login, cmd, port=None, no\_stdin=False*)

Run command on remote machines via SSH.

This is a replacement for a small part of the functionality of SSH. In addition to SSH alone, this command can make use of datalad’s SSH connection management. Its primary use case is to be used with Git as ‘core.sshCommand’ or via “GIT\_SSH\_COMMAND”.

### Parameters

- **login** – [user@]hostname.
- **cmd** – command for remote execution.
- **port** – port to connect to on the remote host. [Default: None]
- **no\_stdin** (*bool, optional*) – Redirect stdin from /dev/null. [Default: False]

## datalad.api.siblings

`datalad.api.siblings` (*action='query', dataset=None, name=None, url=None, pushurl=None, description=None, fetch=False, as\_common\_datasrc=None, publish\_depends=None, publish\_by\_default=None, annex\_wanted=None, annex\_required=None, annex\_group=None, annex\_groupwanted=None, inherit=False, get\_annex\_info=True, recursive=False, recursion\_limit=None*)

Manage sibling configuration

This command offers four different actions: ‘query’, ‘add’, ‘remove’, ‘configure’. ‘query’ is the default action and can be used to obtain information about (all) known siblings. ‘add’ and ‘configure’ are highly similar actions, the only difference being that adding a sibling with a name that is already registered will fail, whereas re-configuring a (different) sibling under a known name will not be considered an error. Lastly, the ‘remove’ action allows for the removal (or de-configuration) of a registered sibling.

For each sibling (added, configured, or queried) all known sibling properties are reported. This includes:

“**name**” Name of the sibling

“**path**” Absolute path of the dataset

“**url**” For regular siblings at minimum a “fetch” URL, possibly also a “pushurl”

Additionally, any further configuration will also be reported using a key that matches that in the Git configuration.



By default, sibling information is rendered as one line per sibling following this scheme:

```
<dataset_path>: <sibling_name>(<+|->) [<access_specification>]
```

where the + and - labels indicate the presence or absence of a remote data annex at a particular remote, and *access\_specification* contains either a URL and/or a type label for the sibling.

### Parameters

- **action** (*{'query', 'add', 'remove', 'configure'}* or *None*, optional) – command action selection (see general documentation). Constraints: list expected, each value must be one of ('query', 'add', 'remove', 'configure'), or value must be *None*. [Default: 'query']
- **dataset** (*Dataset* or *None*, optional) – specify the dataset to configure. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: None]
- **name** (*str* or *None*, optional) – name of the sibling. For sibling removal this option is mandatory, otherwise the hostname part of a given URL is used as a default. This option can be used to limit 'query' to a specific sibling. Constraints: value must be a string, or value must be *None*. [Default: None]
- **url** (*str* or *None*, optional) – the URL of or path to the dataset sibling named by *name*. For recursive operation it is required that a template string for building subdataset sibling URLs is given. List of currently available placeholders: %%NAME the name of the dataset, where slashes are replaced by dashes. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **pushurl** (*str* or *None*, optional) – in case the *url* cannot be used to publish to the dataset sibling, this option specifies a URL to be used instead. If no *url* is given, *pushurl* serves as *url* as well. Constraints: value must be a string, or value must be *None*. [Default: None]
- **description** (*str* or *None*, optional) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. Constraints: value must be a string, or value must be *None*. [Default: None]
- **fetch** (*bool*, optional) – fetch the sibling after configuration. [Default: False]
- **as\_common\_datasrc** – configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto-enabled special remote). [Default: None]
- **publish\_depends** (*list of str* or *None*, optional) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **publish\_by\_default** (*list of str* or *None*, optional) – add a refspec to be published to this sibling by default if nothing specified. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **annex\_wanted** (*str* or *None*, optional) – expression to specify 'wanted' content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-wanted/> for

- more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **annex\_required** (*str* or *None*, *optional*) – expression to specify ‘required’ content for the repository/sibling. See <https://git-annex.branchable.com/git-annex-required/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
  - **annex\_group** (*str* or *None*, *optional*) – expression to specify a group for the repository. See <https://git-annex.branchable.com/git-annex-group/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
  - **annex\_groupwanted** (*str* or *None*, *optional*) – expression for the group-wanted. Makes sense only if `annex_wanted="groupwanted"` and `annex-group` is given too. See <https://git-annex.branchable.com/git-annex-groupwanted/> for more information. Constraints: value must be a string, or value must be *None*. [Default: *None*]
  - **inherit** (*bool*, *optional*) – if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: *False*]
  - **get\_annex\_info** (*bool*, *optional*) – Whether to query all information about the annex configurations of siblings. Can be disabled if speed is a concern. [Default: *True*]
  - **recursive** (*bool*, *optional*) – if set, recurse into potential subdataset. [Default: *False*]
  - **recursion\_limit** (*int* or *None*, *optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘int’, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
  - **result\_renderer** (`{'default', 'json', 'json_pp', 'tailored'}` or *None*, *optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
  - **result\_xfm** (`{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'}` 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item-or-list’). [Default: ‘list’]

## datalad.api.subdatasets

`datalad.api.subdatasets` (*dataset=None, fulfilled=None, recursive=False, recursion\_limit=None, contains=None, bottomup=False, set\_property=None, delete\_property=None*)

Report subdatasets and their properties.

The following properties are reported (if possible) for each matching subdataset record.

“**name**” Name of the subdataset in the parent (often identical with the relative path in the parent dataset)

“**path**” Absolute path to the subdataset

“**parentds**” Absolute path to the parent dataset

“**revision**” SHA1 of the subdataset commit recorded in the parent dataset

“**state**” Condition of the subdataset: ‘clean’, ‘modified’, ‘absent’, ‘conflict’ as reported by *git submodule*

“**revision\_descr**” Output of *git describe* for the subdataset

“**gitmodule\_url**” URL of the subdataset recorded in the parent

“**gitmodule\_<label>**” Any additional configuration property on record.

Performance note: Requesting *bottomup* reporting order, or a particular numerical *recursion\_limit* implies an internal switch to an alternative query implementation for recursive query that is more flexible, but also notably slower (performs one call to Git per dataset versus a single call for all combined).

### Parameters

- **dataset** (*Dataset or None, optional*) – specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path), or value must be *None*. [Default: *None*]
- **fulfilled** (*bool or None, optional*) – if given, must be a boolean flag indicating whether to report either only locally present or absent datasets. By default subdatasets are reported regardless of their status. Constraints: value must be convertible to type *bool*, or value must be *None*. [Default: *None*]
- **recursive** (*bool, optional*) – if set, recurse into potential subdataset. [Default: *False*]
- **recursion\_limit** (*int or None, optional*) – limit recursion into subdataset to the given number of levels. Constraints: value must be convertible to type ‘*int*’, or value must be *None*. [Default: *None*]
- **contains** (*str or None, optional*) – limit report to the subdatasets containing the given path. If a root path of a subdataset is given the last reported dataset will be the subdataset itself. Constraints: value must be a string, or value must be *None*. [Default: *None*]
- **bottomup** (*bool, optional*) – whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down. [Default: *False*]

- **set\_property** (*list of 2-item sequence of str or None, optional*) – Name and value of one or more subdataset properties to be set in the parent dataset’s .gitmodules file. The value can be a Python format() template string wrapped in ‘<>’ (e.g. ‘<{gitmodule\_name}>’). Supported keywords are any item reported in the result properties of this command, plus ‘refds\_relpath’ and ‘refds\_relname’: the relative path of a subdataset with respect to the base dataset of the command call, and, in the latter case, the same string with all directory separators replaced by dashes. Constraints: list expected, each value must be a string, or value must be *None*. [Default: *None*]
- **delete\_property** (*list of str or None, optional*) – Name of one or more subdataset properties to be removed from the parent dataset’s .gitmodules file. Constraints: list expected, each value must be a string, or value must be *None*. [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. Constraints: value must be one of (‘ignore’, ‘continue’, ‘stop’). [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. Constraints: value must be a callable, or value must be *None*. [Default: *None*]
- **result\_renderer** (*{'default', 'json', 'json\_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. Constraints: value must be one of (‘default’, ‘json’, ‘json\_pp’, ‘tailored’), or value must be *None*. [Default: *None*]
- **result\_xfm** (*{'paths', 'relpaths', 'datasets', 'successdatasets-or-none'} 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. Constraints: value must be one of (‘paths’, ‘relpaths’, ‘datasets’, ‘successdatasets-or-none’), or value must be a callable, or value must be *None*. [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. Constraints: value must be one of (‘generator’, ‘list’, ‘item- or-list’). [Default: ‘list’]

## Miscellaneous commands

<code>api.add_archive_content(archive[, annex, ...])</code>	Add content of an archive under git annex control.
<code>api.crawl([path, is_pipeline, is_template, ...])</code>	Crawl online resource to create or update a dataset.
<code>api.crawl_init([args, template, ...])</code>	Initialize crawling configuration
<code>api.test()</code>	Run internal DataLad (unit)tests.

## datalad.api.add\_archive\_content

```
datalad.api.add_archive_content(archive, annex=None, add_archive_leading_dir=False,
                                strip_leading_dirs=False, leading_dirs_depth=None,
                                leading_dirs_consider=None, use_current_dir=False,
                                delete=False, key=False, exclude=None, rename=None,
                                existing='fail', annex_options=None, copy=False, com-
                                mit=True, allow_dirty=False, stats=None, drop_after=False,
                                delete_after=False)
```

Add content of an archive under git annex control.

This results in the files within archive (which should be under annex control itself) added under annex referenc- ing original archive via custom special remotes mechanism

### Example

```
annex-repo$ datalad add-archive-content my_big_tarball.tar.gz
```

#### Parameters

- **archive** (*str*) – archive file or a key (if *key=True* specified). Constraints: value must be a string.
- **annex** – annex instance to use. [Default: None]
- **add\_archive\_leading\_dir** (*bool, optional*) – flag to place extracted content under a directory which would correspond to archive name with suffix stripped. E.g. for archive *example.zip* its content will be extracted under a directory *example/*. [Default: False]
- **strip\_leading\_dirs** (*bool, optional*) – flag to move all files directories up, from how they were stored in an archive, if that one contained a number (possibly more than 1 down) single leading directories. [Default: False]
- **leading\_dirs\_depth** – maximal depth to strip leading directories to. If not specified (None), no limit. [Default: None]
- **leading\_dirs\_consider** (*list of str or None, optional*) – regular expression(s) for directories to consider to strip away. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **use\_current\_dir** (*bool, optional*) – flag to extract archive under the current directory, not the directory where archive is located. Note that it will be of no effect if *key=True* is given. [Default: False]
- **delete** (*bool, optional*) – flag to delete original archive from the filesystem/git in current tree. Note that it will be of no effect if *key=True* is given. [Default: False]
- **key** (*bool, optional*) – flag to signal if provided archive is not actually a filename on its own but an annex key. [Default: False]
- **exclude** (*list of str or None, optional*) – regular expressions for filenames which to exclude from being added to annex. Applied after *–rename* if that one is specified. For exact matching, use anchoring. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **rename** (*list of str or None, optional*) – regular expressions to rename files before being added under git. First letter defines how to split provided string into two parts: Python regular expression (with groups), and replacement string. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]

- **existing** – what operation to perform a file from archive tries to overwrite an existing file with the same name. ‘fail’ (default) leads to RuntimeError exception. ‘overwrite’ silently replaces existing file. ‘archive-suffix’ instructs to add a suffix (prefixed with a ‘-’) matching archive name from which file gets extracted, and if that one present, ‘numeric-suffix’ is in effect in addition, when incremental numeric suffix (prefixed with a ‘.’) is added until no name collision is longer detected. [Default: ‘fail’]
- **annex\_options** (*str or None, optional*) – additional options to pass to git-annex. Constraints: value must be a string, or value must be *None*. [Default: None]
- **copy** (*bool, optional*) – flag to copy the content of the archive instead of moving. [Default: False]
- **commit** (*bool, optional*) – flag to not commit upon completion. [Default: True]
- **allow\_dirty** (*bool, optional*) – flag that operating on a dirty repository (uncommitted or untracked content) is ok. [Default: False]
- **stats** – ActivityStats instance for global tracking. [Default: None]
- **drop\_after** (*bool, optional*) – drop extracted files after adding to annex. [Default: False]
- **delete\_after** (*bool, optional*) – extract under a temporary directory, git-annex add, and delete after. To be used to “index” files within annex without actually creating corresponding files under git. Note that *annex dropunused* would later remove that load. [Default: False]

#### Returns

**Return type** annex

### datalad.api.crawl

`datalad.api.crawl` (*path=None, is\_pipeline=False, is\_template=False, recursive=False, chdir=None*)  
Crawl online resource to create or update a dataset.

#### Examples

\$ `datalad crawl` # within a dataset having `.datalad/crawl/crawl.cfg`

#### Parameters

- **path** (*str or None, optional*) – configuration (or pipeline if `-is-pipeline`) file defining crawling, or a directory of a dataset on which to perform crawling using its standard crawling specification. Constraints: list expected, each value must be a string, or value must be *None*. [Default: None]
- **is\_pipeline** (*bool, optional*) – flag if provided file is a Python script which defines pipeline(). [Default: False]
- **is\_template** (*bool, optional*) – flag if provided value is the name of the template to use. [Default: False]
- **recursive** (*bool, optional*) – flag to crawl subdatasets as well (for now serially). [Default: False]
- **chdir** (*str or None, optional*) – directory to chdir to for crawling. Constraints: value must be a string, or value must be *None*. [Default: None]

## datalad.api.crawl\_init

`datalad.api.crawl_init` (*args=None, template=None, template\_func=None, save=False*)

Initialize crawling configuration

Allows to specify template and function to generate a crawling pipeline

Examples:

```
$ datalad crawl-init --template openfmri --template-func superdataset_pipeline
```

```
$ datalad crawl-init --template fcptable dataset=Baltimore tarballs=True
```

### Parameters

- **args** (*sequence of str or None, optional*) – keyword arguments to pass into the template function generating actual pipeline, organized in a dict. Constraints: value must be a string, or value must be *None*. [Default: None]
- **template** (*str or None, optional*) – the name of the template. Constraints: value must be a string, or value must be *None*. [Default: None]
- **template\_func** – the name of the function. [Default: None]
- **save** (*bool, optional*) – flag to save file into git repo. [Default: False]

## datalad.api.test

`datalad.api.test` ()

Run internal DataLad (unit)tests.

This can be used to verify correct operation on the system

## Support functionality

<code>auto</code>	Proxy basic file operations (e.g.
<code>cmd</code>	Wrapper for command and function calls, allowing for dry runs and output handling
<code>consts</code>	constants for datalad
<code>log</code>	
<code>utils</code>	
<code>version</code>	Defines version to be imported in the module and obtained from setup.py
<code>support.annexrepo</code>	Interface to git-annex by Joey Hess.
<code>support.archives</code>	Various handlers/functionality for different types of files (e.g.
<code>support.configparserinc</code>	
<code>customremotes.main</code>	
<code>customremotes.base</code>	Base classes to custom git-annex remotes (e.g.
<code>customremotes.archives</code>	Custom remote to support getting the load from archives present under annex

## datalad.auto

Proxy basic file operations (e.g. open) to auto-obtain files upon I/O

**class** `datalad.auto.AutomagicIO` (*autoget=True, activate=False*)

Bases: `object`

Class to proxy commonly used API for accessing files so they get automatically fetched

Currently supports builtin `open()` and `h5py.File` when those are read

**activate** ()

**active**

**autoget**

**deactivate** ()

## datalad.cmd

Wrapper for command and function calls, allowing for dry runs and output handling

**class** `datalad.cmd.GitRunner` (*\*args, \*\*kwargs*)

Bases: `datalad.cmd.Runner`

Runner to be used to run git and git annex commands

Overloads the runner class to check & update `GIT_DIR` and `GIT_WORK_TREE` environment variables set to the absolute path if is defined and is relative path

**static** `get_git_environ_adjusted` (*env=None*)

Replaces `GIT_DIR` and `GIT_WORK_TREE` with absolute paths if relative path and defined

**run** (*cmd, env=None, \*args, \*\*kwargs*)

**class** `datalad.cmd.Runner` (*cwd=None, env=None, protocol=None, log\_outputs=None*)

Bases: `object`

Provides a wrapper for calling functions and commands.

An object of this class provides a methods that calls shell commands or python functions, allowing for protocoling the calls and output handling.

Outputs (stdout and stderr) can be either logged or streamed to system's stdout/stderr during execution. This can be enabled or disabled for both of them independently. Additionally, a protocol object can be a used with the Runner. Such a protocol has to implement `datalad.support.protocol.ProtocolInterface`, is able to record calls and allows for dry runs.

**call** (*f, \*args, \*\*kwargs*)

Helper to unify collection of logging all “dry” actions.

Calls *f* if *Runner*-object is not in dry-mode. Adds *f* along with its arguments to *commands* otherwise.

*f*: callable

**\*args, \*\*kwargs**: Callable arguments

**commands**

**cwd**

**dry**

**env**

**log** (*msg, \*args, \*\*kwargs*)

log helper

Logs at DEBUG-level by default and adds “Protocol:”-prefix in order to log the used protocol.



`log_cwd``log_env``log_outputs``log_stdin``protocol`

**run** (*cmd*, *log\_stdout=True*, *log\_stderr=True*, *log\_online=False*, *expect\_stderr=False*, *expect\_fail=False*, *cwd=None*, *env=None*, *shell=None*, *stdin=None*)  
 Runs the command *cmd* using shell.

In case of dry-mode *cmd* is just added to *commands* and it is actually executed otherwise. Allows for separately logging stdout and stderr or streaming it to system's stdout or stderr respectively.

**Note: Using a string as *cmd* and *shell=True* allows for piping**, multiple commands, etc., but that implies `shlex.split()` is not used. This is considered to be a security hazard. So be careful with input.

### Parameters

- **cmd** (*str*, *list*) – String (or list) defining the command call. No shell is used if *cmd* is specified as a list
- **log\_stdout** (*bool*, *optional*) – If True, stdout is logged. Goes to `sys.stdout` otherwise.
- **log\_stderr** (*bool*, *optional*) – If True, stderr is logged. Goes to `sys.stderr` otherwise.
- **log\_online** (*bool*, *optional*) – Either to log as output comes in. Setting to True is preferable for running user-invoked actions to provide timely output
- **expect\_stderr** (*bool*, *optional*) – Normally, having stderr output is a signal of a problem and thus it gets logged at ERROR level. But some utilities, e.g. `wget`, use stderr for their progress output. Whenever such output is expected, set it to True and output will be logged at DEBUG level unless exit status is non-0 (in non-online mode only, in online – would log at DEBUG)
- **expect\_fail** (*bool*, *optional*) – Normally, if command exits with non-0 status, it is considered an ERROR and logged accordingly. But if the call intended for checking routine, such alarming message should not be logged as ERROR, thus it will be logged at DEBUG level.
- **cwd** (*string*, *optional*) – Directory under which run the command (passed to `Popen`)
- **env** (*string*, *optional*) – Custom environment to pass
- **shell** (*bool*, *optional*) – Run command in a shell. If not specified, then it runs in a shell only if command is specified as a string (not a list)
- **stdin** (*file descriptor*) – input stream to connect to stdin of the process.

### Returns

**Return type** (stdout, stderr)

**Raises** `CommandError` – if command's exitcode wasn't 0 or None. `exitcode` is passed to `CommandError`'s `code`-field. Command's stdout and stderr are stored in `CommandError`'s `stdout` and `stderr` fields respectively.

`datalad.cmd.get_runner(*args, **kwargs)`

`datalad.cmd.link_file_load(src, dst, dry_run=False)`  
Just a little helper to hardlink files's load

### datalad.consts

constants for datalad

### datalad.log

**class** `datalad.log.ColorFormatter` (*use\_color=None, log\_name=False, log\_pid=False*)  
Bases: `logging.Formatter`  
**format** (*record*)

### datalad.utils

`datalad.utils.any_re_search(regexes, value)`  
Return if any of regexes (list or str) searches successfully for value

`datalad.utils.assure_bool(s)`  
Convert value into boolean following convention for strings  
to recognize on, True, yes as True, off, False, no as False

`datalad.utils.assure_dict_from_str(s, **kwargs)`  
Given a multiline string with key=value items convert it to a dictionary

#### Parameters

- **s** (*str or dict*) –
- **None if input s is empty** (*Returns*) –

`datalad.utils.assure_dir(*args)`  
Make sure directory exists.

Joins the list of arguments to an os-specific path to the desired directory and creates it, if it not exists yet.

`datalad.utils.assure_list(s, copy=False, iterate=True)`  
Given not a list, would place it into a list. If None - empty list is returned

#### Parameters

- **s** (*list or anything*) –
- **copy** (*bool, optional*) – If list is passed, it would generate a shallow copy of the list
- **iterate** (*bool, optional*) – If it is not a list, but something iterable (but not a `text_type`) iterate over it.

`datalad.utils.assure_list_from_str(s, sep='\n')`  
Given a multiline string convert it to a list of return None if empty

#### Parameters **s** (*str or list*) –

`datalad.utils.assure_tuple_or_list(obj)`  
Given an object, wrap into a tuple if not list or tuple

`datalad.utils.assure_unicode(s, encoding='utf-8')`  
Convert/decode to unicode (PY2) or str (PY3) if of 'binary\_type'

`datalad.utils.auto_repr` (*cls*)

Decorator for a class to assign it an automagic quick and dirty `__repr__`

It uses public class attributes to prepare repr of a class

Original idea: <http://stackoverflow.com/a/27799004/1265472>

`datalad.utils.better_wraps` (*to\_be\_wrapped*)

Decorator to replace `functools.wraps`

This is based on `wrapt` instead of `functools` and in opposition to `wraps` preserves the correct signature of the decorated function. It is written with the intention to replace the use of `wraps` without any need to rewrite the actual decorators.

**class** `datalad.utils.chpwd` (*path*, *mkdir=False*, *logsuffix=''*)

Bases: `object`

Wrapper around `os.chdir` which also adjusts `environ['PWD']`

The reason is that otherwise `PWD` is simply inherited from the shell and we have no ability to assess directory path without dereferencing symlinks.

If used as a context manager it allows to temporarily change directory to the given path

`datalad.utils.encode_filename` (*filename*)

Encode unicode filename

`datalad.utils.escape_filename` (*filename*)

Surround filename in `"` and escape `"` in the filename

`datalad.utils.expandpath` (*path*, *force\_absolute=True*)

Expand all variables and user handles in a path.

By default return an absolute path

`datalad.utils.file_basename` (*name*, *return\_ext=False*)

Strips up to 2 extensions of length up to 4 characters and starting with alpha not a digit, so we could get rid of `.tar.gz` etc

`datalad.utils.find_files` (*regex*, *topdir='.'*, *exclude=None*, *exclude\_vcs=True*, *exclude\_datalad=False*, *dirs=False*)

Generator to find files matching regex

#### Parameters

- **regex** (*basestring*) –
- **exclude** (*basestring*, *optional*) – Matches to exclude
- **exclude\_vcs** – If True, excludes commonly known VCS subdirectories. If string, used as regex to exclude those files (regex: `^(?:git|gitattributes|svn|bzr|hg)(?:/|$)`)
- **exclude\_datalad** – If True, excludes files known to be datalad meta-data files (e.g. under `.datalad/` subdirectory) (regex: `^(?:datalad)(?:/|$)`)
- **topdir** (*basestring*, *optional*) – Directory where to search
- **dirs** (*bool*, *optional*) – Either to match directories as well as files

`datalad.utils.get_dataset_root` (*path*)

Return the root of an existent dataset containing a given path

The root path is returned in the same absolute or relative form as the input argument. If no associated dataset exists, or the input path doesn't exist, `None` is returned.

`datalad.utils.get_func_kwargs_doc(func)`

Provides args for a function

**Parameters** `func` (*str*) – name of the function from which args are being requested

**Returns** of the args that a function takes in

**Return type** list

`datalad.utils.get_logfilename(dspath, cmd='datalad')`

Return a filename to use for logging under a dataset/repository

directory would be created if doesn't exist, but `dspath` must exist and be a directory

`datalad.utils.get_path_prefix(path, pwd=None)`

Get path prefix (for current directory)

Returns relative path to the topdir, if we are under topdir, and if not absolute path to topdir. If `pwd` is not specified - current directory assumed

`datalad.utils.get_tempfile_kwargs(tkwargs=None, prefix='', wrapped=None)`

Updates kwargs to be passed to tempfile. calls depending on env vars

`datalad.utils.get_timestamp_suffix(time_=None, prefix='-')`

Return a time stamp (full date and time up to second)

primarily to be used for generation of log files names

`datalad.utils.get_trace(edges, start, end, trace=None)`

Return the trace/path to reach a node in a tree.

**Parameters**

- **edges** (*sequence (2-tuple)*) – The tree given by a sequence of edges (parent, child) tuples. The nodes can be identified by any value and data type that supports the '==' operation.
- **start** – Identifier of the start node. Must be present as a value in the parent location of an edge tuple in order to be found.
- **end** – Identifier of the target/end node. Must be present as a value in the child location of an edge tuple in order to be found.
- **trace** (*list*) – Mostly useful for recursive calls, and used internally.

**Returns** Returns a list with the trace to the target (the starts and the target are not included in the trace, hence if start and end are directly connected an empty list is returned), or None when no trace to the target can be found, or start and end are identical.

**Return type** None or list

`datalad.utils.getpwd()`

Try to return a CWD without dereferencing possible symlinks

If no PWD found in the env, output of `getcwd()` is returned

`datalad.utils.is_explicit_path(path)`

Return whether a path explicitly points to a location

Any absolute path, or relative path starting with either `'..'` or `'.'` is assumed to indicate a location on the filesystem. Any other path format is not considered explicit.

`datalad.utils.is_interactive()`

Return True if all in/outs are tty

`datalad.utils.knows_annex` (*path*)

Returns whether at a given path there is information about an annex

It is just a thin wrapper around `GitRepo.is_with_annex()` classmethod which also checks for *path* to exist first.

This includes actually present annexes, but also uninitialized ones, or even the presence of a remote annex branch.

`datalad.utils.line_profile` (*func*)

`datalad.utils.lmtime` (*filepath*, *mtime*)

Set *mtime* for files, while not de-referencing symlinks.

To overcome absence of `os.lutime`

Works only on linux and OSX ATM

`datalad.utils.make_tempfile` (*\*args*, *\*\*kwargs*)

Helper class to provide a temporary file name and remove it at the end (context manager)

#### Parameters

- **mkdir** (*bool*, *optional* (default: `False`)) – If True, temporary directory created using `tempfile.mkdtemp()`
- **content** (*str* or *bytes*, *optional*) – Content to be stored in the file created
- **wrapped** (*function*, *optional*) – If set, function name used to prefix temporary file name
- **\*\*kwargs** – All other arguments are passed into the call to `tempfile.mk{,d}temp()`, and resultant temporary filename is passed as the first argument into the function *t*. If no ‘prefix’ argument is provided, it will be constructed using module and function names (‘.’ replaced with ‘\_’).
- **change the used directory without providing keyword argument 'dir' set (To) –**
- `DATALAD_TESTS_TEMP_DIR.` –

#### Examples

```
>>> from os.path import exists
>>> from datalad.utils import make_tempfile
>>> with make_tempfile() as fname:
...     k = open(fname, 'w').write('silly test')
>>> assert not exists(fname) # was removed
```

```
>>> with make_tempfile(content="blah") as fname:
...     assert open(fname).read() == "blah"
```

`datalad.utils.md5sum` (*filename*)

`datalad.utils.not_supported_on_windows` (*msg=None*)

A little helper to be invoked to consistently fail whenever functionality is not supported (yet) on Windows

`datalad.utils.nothing_cm` (*\*args*, *\*\*kwargs*)

Just a dummy cm to programmatically switch context managers

`datalad.utils.optional_args` (*decorator*)

allows a decorator to take optional positional and keyword arguments. Assumes that taking a single, callable, positional argument means that it is decorating a function, i.e. something like this:

```
@my_decorator
def function(): pass
```

Calls decorator with `decorator(f, *args, **kwargs)`

`datalad.utils.posix_relpath` (*path, start=None*)

Behave like `os.path.relpath`, but always return POSIX paths...

on any platform.

`datalad.utils.rmtemp` (*f, \*args, \*\*kwargs*)

Wrapper to centralize removing of temp files so we could keep them around

It will not remove the temporary file/directory if `DATALAD_TESTS_TEMP_KEEP` environment variable is defined

`datalad.utils.rmtree` (*path, chmod\_files='auto', \*args, \*\*kwargs*)

To remove git-annex `.git` it is needed to make all files and directories writable again first

#### Parameters

- **chmod\_files** (*string or bool, optional*) – Either to make files writable also before removal. Usually it is just a matter of directories to have write permissions. If ‘auto’ it would chmod files on windows by default
- **\*args** –
- **\*\*kwargs** – Passed into `shutil.rmtree` call

`datalad.utils.rotree` (*path, ro=True, chmod\_files=True*)

To make tree read-only or writable

#### Parameters

- **path** (*string*) – Path to the tree/directory to chmod
- **ro** (*bool, optional*) – Either to make it R/O (default) or RW
- **chmod\_files** (*bool, optional*) – Either to operate also on files (not just directories)

`datalad.utils.saved_generator` (*gen*)

Given a generator returns two generators, where 2nd one just replays

So the first one would be going through the generated items and 2nd one would be yielding saved items

`datalad.utils.setup_exceptionhook` (*ipython=False*)

Overloads default `sys.excepthook` with our exceptionhook handler.

If interactive, our exceptionhook handler will invoke `pdb.post_mortem`; if not interactive, then invokes default handler.

`datalad.utils.shortened_repr` (*value, l=30*)

`datalad.utils.slash_join` (*base, extension*)

Join two strings with a ‘/’, avoiding duplicate slashes

If any of the strings is `None` the other is returned as is.

`datalad.utils.sorted_files` (*dout*)

Return a (sorted) list of files under `dout`

`datalad.utils.swallow_logs(*args, **kws)`

Context manager to consume all logs.

`datalad.utils.swallow_outputs(*args, **kws)`

Context manager to help consuming both stdout and stderr, and print()

stdout is available as `cm.out` and stderr as `cm.err` whenever `cm` is the yielded context manager. Internally uses temporary files to guarantee absent side-effects of swallowing into `StringIO` which lacks `.fileno`.

print mocking is necessary for some uses where `sys.stdout` was already bound to original `sys.stdout`, thus mocking it later had no effect. Overriding print function had desired effect

`datalad.utils.try_multiple(ntrials, exception, base, f, *args, **kwargs)`

Call `f` multiple times making exponentially growing delay between the calls

`datalad.utils.unique(seq, key=None)`

Given a sequence return a list only with unique elements while maintaining order

This is the fastest solution. See <https://www.peterbe.com/plog/uniqifiers-benchmark> and <http://stackoverflow.com/a/480227/1265472> for more information. Enhancement – added ability to compare for uniqueness using a key function

#### Parameters

- **seq** – Sequence to analyze
- **key** (*callable, optional*) – Function to call on each element so we could decide not on a full element, but on its member etc

`datalad.utils.updated(d, update)`

Return a copy of the input with the ‘update’

Primarily for updating dictionaries

`datalad.utils.with_pathsep(path)`

Little helper to guarantee that path ends with `/`

## datalad.version

Defines version to be imported in the module and obtained from `setup.py`

## datalad.support.annexrepo

Interface to git-annex by Joey Hess.

For further information on git-annex see <https://git-annex.branchable.com/>.

```
class datalad.support.annexrepo.AnnexRepo(path, url=None, runner=None, direct=None,
                                         backend=None, always_commit=True, create=True,
                                         init=False, batch_size=None, version=None,
                                         description=None, git_opts=None, annex_opts=None,
                                         annex_init_opts=None, repo=None)
```

Bases: `datalad.support.gitrepo.GitRepo`, `datalad.support.repo.RepoInterface`

Representation of an git-annex repository.

Paths given to any of the class methods will be interpreted as relative to PWD, in case this is currently beneath `AnnexRepo`’s base dir (*self.path*). If PWD is outside of the repository, relative paths will be interpreted as relative to *self.path*. Absolute paths will be accepted either way.

`GIT_ANNEX_MIN_VERSION = '6.20170220'`

`WEB_UUID = '00000000-0000-0000-0000-000000000001'`

`add(files, *args, **kwargs)`

Add file(s) to the repository.

#### Parameters

- **files** (*list of str*) – list of paths to add to the annex
- **git** (*bool*) – if True, add to git instead of annex.
- **commit** (*bool*) – whether or not to directly commit
- **msg** (*str*) – commit message in case *commit=True*. A default message, containing the list of files that were added, is created by default.
- **backend** –
- **options** –
- **dry\_run** (*bool, optional*) – Calls git add with `-dry-run -N -ignore-missing`, to just output list of files to be added
- **update** (*bool*) –

**–update option for git-add. From git’s manpage:** Update the index just where it already has an entry matching `<pathspec>`. This removes as well as modifies index entries to match the working tree, but adds no new files.

If no `<pathspec>` is given when `–update` option is used, all tracked files in the entire working tree are updated (old versions of Git used to limit the update to the current directory and its subdirectories).

Note: Used only, if a call to git-add instead of git-annex-add is performed

#### Returns

**Return type** list of dict

`add_remote(name, url, options=None)`

Overrides method from GitRepo in order to set `remote.<name>.annex-ssh-options` in case of a SSH remote.

`add_url_to_file(file_, *args, **kwargs)`

Add file from url to the annex.

Downloads *file* from *url* and add it to the annex. If annex knows *file* already, records that it can be downloaded from *url*.

#### Parameters

- **file** (*str*) –
- **url** (*str*) –
- **options** (*list*) – options to the annex command
- **batch** (*bool, optional*) – initiate or continue with a batched run of annex addurl, instead of just calling a single git annex addurl command
- **unlink\_existing** (*bool, optional*) – by default crashes if file already exists and is under git. With this flag set to True would first remove it.

**Returns** In batch mode only ATM returns dict representation of json output returned by annex

**Return type** dict



**add\_urls** (*urls*, *options=None*, *backend=None*, *cwd=None*, *jobs=None*, *git\_options=None*, *annex\_options=None*)

Downloads each url to its own file, which is added to the annex.

**Parameters**

- **urls** (*list of str*) –
- **options** (*list, optional*) – options to the annex command
- **cwd** (*string, optional*) – working directory from within which to invoke git-annex

**adjust** (*options=None*)

enter an adjusted branch

This command is only available in a v6 git-annex repository.

**Parameters** **options** (*list of str*) – currently requires ‘–unlock’ or ‘–fix’; default: –unlock

**commit** (*msg=None*, *options=None*, *\_datalad\_msg=False*, *careless=True*, *files=None*, *proxy=False*)

Commit changes to git.

**Parameters**

- **msg** (*str, optional*) – commit-message
- **options** (*list of str, optional*) – cmdline options for git-commit
- **\_datalad\_msg** (*bool, optional*) – To signal that commit is automated commit by datalad, so it would carry the [DATALAD] prefix
- **careless** (*bool, optional*) – if False, raise when there’s nothing actually committed; if True, don’t care
- **files** (*list of str, optional*) – path(s) to commit

**copy\_to** (*files, \*args, \*\*kwargs*)

Copy the actual content of *files* to *remote*

**Parameters**

- **files** (*str or list of str*) – path(s) to copy
- **remote** (*str*) – name of remote to copy *files* to
- **log\_online** (*bool*) – see get()

**Returns** files successfully copied

**Return type** list of str

**default\_backends**

**drop** (*files, \*args, \*\*kwargs*)

Drops the content of annexed files from this repository.

Drops only if possible with respect to required minimal number of available copies.

**Parameters**

- **files** (*list of str*) – paths to drop
- **options** (*list of str, optional*) – commandline options for the git annex drop command
- **jobs** (*int, optional*) – how many jobs to run in parallel (passed to git-annex call)

**Returns** ‘success’ item in each object indicates failure/success per file path.

**Return type** list(JSON objects)

**drop\_key** (*keys*, *options=None*, *batch=False*)

Drops the content of annexed files from this repository referenced by keys

Dangerous: it drops without checking for required minimal number of available copies.

**Parameters**

- **keys** (*list of str*, *str*) –
- **batch** (*bool*, *optional*) – initiate or continue with a batched run of annex dropkey, instead of just calling a single git annex dropkey command

**enable\_remote** (*name*)

Enables use of an existing special remote

**Parameters** **name** (*str*) – name, the special remote was created with

**file\_has\_content** (*files*, *\*args*, *\*\*kwargs*)

Check whether files have their content present under annex.

**Parameters**

- **files** (*list of str*) – file(s) to check for being actually present.
- **allow\_quick** (*bool*, *optional*) – allow quick check, based on having a symlink into .git/annex/objects. Works only in non-direct mode (TODO: thin mode)

**Returns** For each input file states either file has content locally

**Return type** list of bool

**find** (*files*, *\*args*, *\*\*kwargs*)

Provide annex info for file(s).

**Parameters**

- **files** (*list of str*) – files to find under annex
- **batch** (*bool*, *optional*) – initiate or continue with a batched run of annex find, instead of just calling a single git annex find command

**Returns** list with filename if file found else empty string

**Return type** list

**fsck** ()

**get** (*files*, *\*args*, *\*\*kwargs*)

Get the actual content of files

**Parameters**

- **files** (*list of str*) – paths to get
- **options** (*list of str*, *optional*) – commandline options for the git annex get command
- **jobs** (*int*, *optional*) – how many jobs to run in parallel (passed to git-annex call)

**Returns** files

**Return type** list of dict

**get\_annexed\_files** (*with\_content\_only=False*)

Get a list of files in annex

**get\_contentlocation** (*key*, *batch=False*)

Get location of the key content

Normally under .git/annex objects in indirect mode and within file tree in direct mode.

Unfortunately there is no (easy) way to discriminate situations when given key is simply incorrect (not known to annex) or its content not currently present – in both cases annex just silently exits with -1

**Parameters**

- **key** (*str*) – key
- **batch** (*bool*, *optional*) – initiate or continue with a batched run of annex content-location

**Returns** path relative to the top directory of the repository. If no content is present, empty string is returned

**Return type** *str*

**get\_corresponding\_branch** (*branch=None*)

In case of a managed branch, get the corresponding one.

If *branch* is not a managed branch, return that branch without any changes.

Note: Since default for *branch* is the active branch, *get\_corresponding\_branch()* is equivalent to *get\_active\_branch()* if the active branch is not a managed branch.

**Parameters** **branch** (*str*) – name of the branch; defaults to active branch

**Returns** name of the corresponding branch if there is any, name of the queried branch otherwise.

**Return type** *str*

**get\_description** (*uuid=None*)

Get annex repository description

**Parameters** **uuid** (*str*, *optional*) – For which remote (based on uuid) to report description for

**Returns** None returned if not found

**Return type** *str* or *None*

**get\_file\_backend** (*files*, *\*args*, *\*\*kwargs*)

Get the backend currently used for file(s).

**Parameters** **files** (*list of str*) –

**Returns** For each file in input list indicates the used backend by a str like “SHA256E” or “MD5”.

**Return type** list of *str*

**get\_file\_key** (*files*, *\*args*, *\*\*kwargs*)

Get key of an annexed file.

**Parameters** **files** (*str or list*) – file(s) to look up

**Returns** keys used by git-annex for each of the files; in case of a list an empty string is returned if there was no key for that file

**Return type** *str* or list

**get\_file\_size** (*file\_*, *\*args*, *\*\*kwargs*)

**get\_groupwanted** (*name*)

Get *groupwanted* expression for a group *name*

**Parameters** **name** (*str*) – Name of the group/wanted group

**get\_metadata** (*files, timestamps=False*)

Query git-annex file metadata

**Parameters**

- **files** (*str* or *list(str)*) – One or more paths for which metadata is to be queried.
- **timestamps** (*bool, optional*) – If True, the output contains a ‘<metadatakey>-lastchanged’ key for every metadata item, reflecting the modification time, as well as a ‘lastchanged’ key with the most recent modification time of any metadata item.

**Returns** One tuple per file (could be more items than input arguments when directories are given). First tuple item is the filename, second item is a dictionary with metadata key/value pairs. Note that annex metadata tags are stored under the key ‘tag’, which is a regular metadata item that can be manipulated like any other.

**Return type** generator

**get\_preferred\_content** (*property, remote=None*)

Get preferred content configuration of a repository or remote

**Parameters**

- **property** (*{'wanted', 'required', 'group'}*) – Type of property to query
- **remote** (*str, optional*) – If not specified (None), returns the property for the local repository.

**Returns** Either the setting is returned, or an empty string if there is none.

**Return type** *str*

**Raises**

- `ValueError` – If an unknown property label is given.
- `CommandError` – If the annex call errors.

**get\_remotes** (*with\_refs\_only=False, with\_urls\_only=False, exclude\_special\_remotes=False*)

Get known (special-) remotes of the repository

**Parameters**

- **exclude\_special\_remotes** (*bool, optional*) – if True, don’t return annex special remotes
- **with\_refs\_only** (*bool, optional*) – return only remotes with any refs. E.g. annex special remotes would not have any refs

**Returns** **remotes** – List of names of the remotes

**Return type** list of *str*

**static get\_size\_from\_key** (*key*)

A little helper to obtain size encoded in a key

**get\_status** (*untracked=True, deleted=True, modified=True, added=True, type\_changed=True, submodules=True, path=None*)

Return various aspects of the status of the annex repository

Note: Under certain circumstances newly added submodules might be reported as ‘modified’ rather than ‘added’. See `AnnexRepo._submodules_dirty_direct_mode` for details.

**Parameters**

- `untracked` –
- `deleted` –
- `modified` –
- `added` –
- `type_changed` –
- `submodules` –
- `path` –

**classmethod** `get_toppath` (*path*, *follow\_up=True*, *git\_options=None*)

Return top-level of a repository given the path.

**Parameters**

- **follow\_up** (*bool*) – If path has symlinks – they get resolved by git. If `follow_up` is `True`, we will follow original path up until we hit the same resolved path. If no such path found, resolved one would be returned.
- **git\_options** (*list of str*) – options to be passed to the git rev-parse call
- **None if no parent directory contains a git repository.**  
(*Return*) –

**get\_tracking\_branch** (*branch=None*, *corresponding=True*)

Get the tracking branch for *branch* if there is any.

By default returns the tracking branch of the corresponding branch if *branch* is a managed branch.

**Parameters**

- **branch** (*str*) – local branch to look up. If none is given, active branch is used.
- **corresponding** (*bool*) – If `True` actually look up the corresponding branch of *branch* (also if *branch* isn't explicitly given)

**Returns** (remote or `None`, refspec or `None`) of the tracking branch

**Return type** `tuple`

**get\_urls** (*file\_*, *\*args*, *\*\*kwargs*)

Get URLs for a file/key

**Parameters**

- **file** (*str*) –
- **key** (*bool*, *optional*) – Either provided files are actually annex keys

**git\_annex\_version = None**

**info** (*files*, *\*args*, *\*\*kwargs*)

Provide annex info for file(s).

**Parameters** **files** (*list of str*) – files to look for

**Returns** Info for each file

**Return type** `dict`

**init\_remote** (*name*, *options*)

Creates a new special remote

**Parameters** **name** (*str*) – name of the special remote

**is\_available** (*files, \*args, \*\*kwargs*)

Check if file or key is available (from a remote)

In case if key or remote is misspecified, it wouldn't fail but just keep returning False, although possibly also complaining out loud ;)

**Parameters**

- **file** (*str*) – Filename or a key
- **remote** (*str, optional*) – Remote which to check. If None, possibly multiple remotes are checked before positive result is reported
- **key** (*bool, optional*) – Either provided files are actually annex keys
- **batch** (*bool, optional*) – Initiate or continue with a batched run of annex checkpresentkey

**Returns** with True indicating that file/key is available from (the) remote

**Return type** `bool`

**is\_crippled\_fs** ()

Return True if git-annex considers current filesystem 'crippled'.

**Returns**

**Return type** True if on crippled filesystem, False otherwise

**is\_direct\_mode** ()

Return True if annex is in direct mode

**Returns**

**Return type** True if in direct mode, False otherwise.

**is\_dirty** (*index=True, working\_tree=False, untracked\_files=True, submodules=True, path=None*)

Returns true if the repo is considered to be dirty

**Parameters**

- **index** (*bool*) – if True, consider changes to the index
- **working\_tree** (*bool*) – if True, consider changes to the working tree
- **untracked\_files** (*bool*) – if True, consider untracked files
- **submodules** (*bool*) – if True, consider submodules
- **path** (*str or list of str*) – path(s) to consider only

**Returns**

**Return type** `bool`

**is\_managed\_branch** (*branch=None*)

Whether *branch* is managed by git-annex.

ATM this returns true in direct mode (branch 'annex/direct/my\_branch') and if on an adjusted branch (annex v6 repository: either 'adjusted/my\_branch(unlocked)' or 'adjusted/my\_branch(fixed)')

Note: The term 'managed branch' is used to make clear it's meant to be more general than the v6 'adjusted branch'.

**Parameters** **branch** (*str*) – name of the branch; default: active branch

**Returns** True if on a managed branch, False otherwise

**Return type** bool

**is\_under\_annex** (*files*, \*args, \*\*kwargs)

Check whether files are under annex control

**Parameters**

- **files** (*list of str*) – file(s) to check for being under annex
- **allow\_quick** (*bool, optional*) – allow quick check, based on having a symlink into .git/annex/objects. Works only in non-direct mode (TODO: thin mode)

**Returns** For each input file states either file is under annex

**Return type** list of bool

**classmethod is\_valid\_repo** (*path*, *allow\_noninitialized=False*)

Return True if given path points to an annex repository

**lock** (*files*, \*args, \*\*kwargs)

undo unlock

Use this to undo an unlock command if you don't want to modify the files any longer, or have made modifications you want to discard.

**Parameters**

- **files** (*list of str*) –
- **options** (*list of str*) –

**merge\_annex** (*remote=None*)

Merge git-annex branch

Merely calls *sync* with the appropriate arguments.

**Parameters** **remote** (*str, optional*) – Name of a remote to be “merged”.

**migrate\_backend** (*files*, \*args, \*\*kwargs)

Changes the backend used for *file*.

The backend used for the key-value of *files*. Only files currently present are migrated. Note: There will be no notification if migrating fails due to the absence of a file's content!

**Parameters**

- **files** (*list*) – files to migrate.
- **backend** (*str*) – specify the backend to migrate to. If none is given, the default backend of this instance will be used.

**precommit** ()

Perform pre-commit maintenance tasks, such as closing all batched annexes since they might still need to flush their changes into index

**proxy** (*git\_cmd*, \*\*kwargs)

Use git-annex as a proxy to git

This is needed in case we are in direct mode, since there's no git working tree, that git can handle.

**Parameters**

- **git\_cmd** (*list of str*) – the actual git command
- **\*\*kwargs** (*dict, optional*) – passed to `_run_annex_command`

**Returns** output of the command call

**Return type** (stdout, stderr)

**remove** (*files*, \*args, \*\*kwargs)

Remove files from git/annex (works in direct mode as well)

**Parameters**

- **files** –
- **force** (*bool*, *optional*) –

**repo\_info** (*fast=False*)

Provide annex info for the entire repository.

**Returns** Info for the repository, with keys matching the ones returned by annex

**Return type** dict

**rm\_url** (*file\_*, \*args, \*\*kwargs)

Record that the file is no longer available at the url.

**Parameters**

- **file** (*str*) –
- **url** (*str*) –

**set\_direct\_mode** (*enable\_direct\_mode=True*)

Switch to direct or indirect mode

**Parameters** **enable\_direct\_mode** (*bool*) – True means switch to direct mode, False switches to indirect mode

**Raises** `CommandNotAvailableError` – in case you try to switch to indirect mode on a crippled filesystem

**set\_groupwanted** (*name*, *expr*)

Set *expr* for the *name* groupwanted

**set\_metadata** (*files*, *reset=None*, *add=None*, *init=None*, *remove=None*, *purge=None*, *recursive=False*)

Manipulate git-annex file-metadata

**Parameters**

- **files** (*str* or *list(str)*) – One or more paths for which metadata is to be manipulated. The changes applied to each file item are uniform. However, the result may not be uniform across files, depending on the actual operation.
- **reset** (*dict*, *optional*) – Metadata items matching keys in the given dict are (re)set to the respective values.
- **add** (*dict*, *optional*) – The values of matching keys in the given dict appended to any possibly existing values. The metadata keys need not necessarily exist before.
- **init** (*dict*, *optional*) – Metadata items for the keys in the given dict are set to the respective values, if the key is not yet present in a file's metadata.
- **remove** (*dict*, *optional*) – Values in the given dict are removed from the metadata items matching the respective key, if they exist in a file's metadata. Non-existing values, or keys do not lead to failure.
- **purge** (*list*, *optional*) – Any metadata item with a key matching an entry in the given list is removed from the metadata.



- **recursive** (*bool*, *optional*) – If False, fail (with `CommandError`) when directory paths are given as *files*.

**Returns** JSON obj per modified file

**Return type** generator

**set\_preferred\_content** (*property*, *expr*, *remote=None*)

Set preferred content configuration of a repository or remote

**Parameters**

- **property** (`{ 'wanted', 'required', 'group' }`) – Type of property to query
- **expr** (*str*) – Any expression or label supported by git-annex for the given property.
- **remote** (*str*, *optional*) – If not specified (`None`), sets the property for the local repository.

**Returns** Raw git-annex output in response to the set command.

**Return type** *str*

**Raises**

- `ValueError` – If an unknown property label is given.
- `CommandError` – If the annex call errors.

**set\_remote\_url** (*name*, *url*, *push=False*)

Set the URL a remote is pointing to

Sets the URL of the remote *name*. Requires the remote to already exist.

**Parameters**

- **name** (*str*) – name of the remote
- **url** (*str*) –
- **push** (*bool*) – if True, set the push URL, otherwise the fetch URL; if True, additionally set `annexurl` to *url*, to make sure annex uses it to talk to the remote, since access via fetch URL might be restricted.

**sync** (*remotes=None*, *push=True*, *pull=True*, *commit=True*, *content=False*, *all=False*, *fast=False*)

Synchronize local repository with remotes

Use this command when you want to synchronize the local repository with one or more of its remotes. You can specify the remotes (or remote groups) to sync with by name; the default if none are specified is to sync with all remotes.

**Parameters**

- **remotes** (*str*, *list(str)*, *optional*) – Name of one or more remotes to be sync'ed.
- **push** (*bool*) – By default, git pushes to remotes.
- **pull** (*bool*) – By default, git pulls from remotes
- **commit** (*bool*) – A commit is done by default. Disable to avoid committing local changes.
- **content** (*bool*) – Normally, syncing does not transfer the contents of annexed files. This option causes the content of files in the work tree to also be uploaded and downloaded as necessary.

- **all** (*bool*) – This option, when combined with *content*, makes all available versions of all files be synced, when preferred content settings allow
- **fast** (*bool*) – Only sync with the remotes with the lowest annex-cost value configured

**unannex** (*files, \*args, \*\*kwargs*)  
undo accidental add command

Use this to undo an accidental git annex add command. Note that for safety, the content of the file remains in the annex, until you use git annex unused and git annex dropunused.

**Parameters**

- **files** (*list of str*)–
- **options** (*list of str*)–

**Returns** successfully unannexed files

**Return type** list of str

**unlock** (*files, \*args, \*\*kwargs*)  
unlock files for modification

**Parameters**

- **files** (*list of str*)–
- **options** (*list of str*)–

**Returns** successfully unlocked files

**Return type** list of str

**untracked\_files**  
Get a list of untracked files

**uuid**  
Annex UUID

**Returns** Returns a the annex UUID, if there is any, or *None* otherwise.

**Return type** str

**whereis** (*files, \*args, \*\*kwargs*)  
Lists repositories that have actual content of file(s).

**Parameters**

- **files** (*list of str*)– files to look for
- **output** (*{'descriptions', 'uuids', 'full'}, optional*)– If ‘descriptions’, a list of remotes descriptions returned is per each file. If ‘full’, for each file a dictionary of all fields is returned as returned by annex
- **key** (*bool, optional*)– Either provided files are actually annex keys
- **options** (*list, optional*)– Options to pass into git-annex call

**Returns**

if output == ‘descriptions’, contains a list of descriptions of remotes for each input file, describing the remote for each remote, which was found by git-annex whereis, like:

```
u'me@mycomputer:~/where/my/repo/is [origin]' or
u'web' or
u'me@mycomputer:~/some/other/clone'
```

if `output == 'uuids'`, returns a list of uuids. if `output == 'full'`, returns a dictionary with filenames as keys and values a detailed record, e.g.:

```
{'00000000-0000-0000-0000-000000000001': {
  'description': 'web',
  'here': False,
  'urls': ['http://127.0.0.1:43442/about.txt', 'http://example.com/
↪someurl']
}}
```

**Return type** list of list of unicode or dict

**class** `datalad.support.annexrepo.BatchedAnnex` (*annex\_cmd*, *git\_options=None*, *annex\_options=None*, *path=None*, *json=False*, *output\_proc=None*)

Bases: `object`

Container for an annex process which would allow for persistent communication

**close** ()

Close communication and wait for process to terminate

**class** `datalad.support.annexrepo.BatchedAnnexes` (*batch\_size=0*)

Bases: `dict`

Class to contain the registry of active batch'ed instances of annex for a repository

**clear** ()

Override just to make sure we don't rely on `__del__` to close all the pipes

**close** ()

Close communication to all the batched annexes

It does not remove them from the dictionary though

**get** (*codename*, *annex\_cmd=None*, *\*\*kwargs*)

**class** `datalad.support.annexrepo.ProcessAnnexProgressIndicators` (*expected=None*)

Bases: `object`

'Filter' for annex -json output to react to progress indicators

Instance of this beast should be passed into `log_stdout` option for git-annex commands runner

**finish** ()

**start** ()

`datalad.support.annexrepo.readline_json` (*stdout*)

`datalad.support.annexrepo.readline_rstripped` (*stdout*)

`datalad.support.annexrepo.readlines_until_ok_or_failed` (*stdout*, *maxlines=100*)

Read stdout until line ends with ok or failed

## datalad.support.archives

Various handlers/functionality for different types of files (e.g. for archives)

**class** `datalad.support.archives.ArchivesCache` (*toppath=None*, *persistent=False*)

Bases: `object`

Cache to maintain extracted archives

**Parameters**

- **toppath** (*str*) – Top directory under `.git/` of which temp directory would be created. If not provided – random tempdir is used
- **persistent** (*bool*, *optional*) – Passed over into generated `ExtractedArchives`

**clean** (*force=False*)

**get\_archive** (*archive*)

**path**

**class** `datalad.support.archives.ExtractedArchive` (*archive*, *path=None*, *persistent=False*)

Bases: `object`

Container for the extracted archive

**STAMP\_SUFFIX** = `‘.stamp’`

**assure\_extracted** ()

Return path to the extracted *archive*. Extract archive if necessary

**clean** (*force=False*)

**get\_extracted\_file** (*afile*)

**get\_extracted\_filename** (*afile*)

Return full path to the *afile* within extracted *archive*

It does not actually extract any archive

**get\_extracted\_files** ()

Generator to provide filenames which are available under extracted archive

**get\_leading\_directory** (*depth=None*, *consider=None*, *exclude=None*)

Return leading directory of the content within archive

**Parameters**

- **depth** (*int* or *None*, *optional*) – Maximal depth of leading directories to consider. If *None* - no upper limit
- **consider** (*list of str*, *optional*) – Regular expressions for file/directory names to be considered (before *exclude*). Applied to the entire relative path to the file as in the archive
- **exclude** (*list of str*, *optional*) – Regular expressions for file/directory names to be excluded from consideration. Applied to the entire relative path to the file as in the archive

**Returns** If there is no single leading directory – *None* returned

**Return type** *str* or *None*

**is\_extracted**

**path**

Given an archive – return full path to it within cache (extracted)

**stamp\_path**

`datalad.support.archives.compress_files` (*files*, *archive*, *path=None*, *overwrite=True*)

Compress *files* into an *archive* file

**Parameters**

- **files** (*list of str*) –
- **archive** (*str*) –
- **path** (*str*) – Alternative directory under which compressor will be invoked, to e.g. take into account relative paths of files and/or archive
- **overwrite** (*bool*) – Either to allow overwriting the target archive file if one already exists

`datalad.support.archives.decompress_file` (*archive, dir\_, leading\_directories='strip'*)  
Decompress *archive* into a directory *dir\_*

#### Parameters

- **archive** (*str*) –
- **dir** (*str*) –
- **leading\_directories** (*{'strip', None}*) – If *strip*, and archive contains a single leading directory under which all content is stored, all the content will be moved one directory up and that leading directory will be removed.

`datalad.support.archives.unixify_path` (*path*)  
On windows convert paths from drive:d file to /drive/d/file

This overcomes problems with various cmdline tools we are to use, such as tar etc

## datalad.support.configparserinc

`class datalad.support.configparserinc.SafeConfigParserWithIncludes` (*defaults=None, dict\_type=<class 'collections.OrderedDict'>, allow\_no\_value=False*)

Bases: `ConfigParser.SafeConfigParser`

Class adds functionality to `SafeConfigParser` to handle included other configuration files (or may be urls, whatever in the future)

File should have section [includes] and only 2 options implemented are 'files\_before' and 'files\_after' where files are listed 1 per line.

Example:

```
[INCLUDES]
before = 1.conf
        3.conf

after = 1.conf
```

It is a simple implementation, so just basic care is taken about recursion. Includes preserve right order, ie new files are inserted to the list of read configs before original, and their includes correspondingly so the list should follow the leaves of the tree.

I wasn't sure what would be the right way to implement generic (aka c++ template) so we could base at any `*configparser` class... so I will leave it for the future

**SECTION\_NAME = 'INCLUDES'**

**static getIncludes** (*resource*, *seen=[]*)

Given 1 config resource returns list of included files (recursively) with the original one as well Simple loops are taken care about

**read** (*filenames*)

### datalad.customremotes.main

datalad.customremotes.main.**main** (*args=None*, *backend=None*)

datalad.customremotes.main.**setup\_parser** (*backend*)

### datalad.customremotes.base

Base classes to custom git-annex remotes (e.g. extraction from archives)

**class** datalad.customremotes.base.**AnnexCustomRemote** (*path=None*, *cost=None*)

Bases: `object`

Base class to provide custom special remotes for git-annex

Implements git-annex special custom remotes protocol described at [http://git-annex.branchable.com/design/external\\_special\\_remote\\_protocol/](http://git-annex.branchable.com/design/external_special_remote_protocol/)

**AVAILABILITY = 'local'**

**COST = 100**

**CUSTOM\_REMOTE\_NAME = None**

**debug** (*msg*)

**error** (*msg*, *annex\_err='ERROR'*)

**get\_DIRHASH** (*key*, *full=False*)

Gets a two level hash associated with a Key.

#### Parameters

- **full** (*bool*, *optional*) – If True, would spit out full DIRHASH path, i.e. with a KEY/ directory
- **like "abc/def". This is always the same for any given Key, so** (*Something*) –
- **be used for eg, creating hash directory structures to store Keys in.** (*can*) –

**get\_URLS** (*key*)

Gets URL(s) associated with a Key.

**get\_contentlocation** (*key*, *absolute=False*, *verify\_exists=True*)

Return (relative to top or absolute) path to the file containing the key

This is a wrapper around `AnnexRepo.get_contentlocation` which provides caching of the result (we are asking the location for the same archive key often)

**heavydebug** (*msg*, *\*args*, *\*\*kwargs*)

**main** ()

Interface to the command line tool

**progress** (*bytes*)

**read** (*req=None, n=1*)

Read a message from git-annex

#### Parameters

- **req** (*string, optional*) – Expected request - first msg of the response
- **n** (*int*) – Number of response elements after first msg

**req\_CHECKPRESENT** (*key*)

**CHECKPRESENT-SUCCESS Key** Indicates that a key has been positively verified to be present in the remote.

**CHECKPRESENT-FAILURE Key** Indicates that a key has been positively verified to not be present in the remote.

**CHECKPRESENT-UNKNOWN Key ErrorMessage** Indicates that it is not currently possible to verify if the key is present in the remote. (Perhaps the remote cannot be contacted.)

**req\_CHECKURL** (*url*)

The remote replies with one of CHECKURL-FAILURE, CHECKURL-CONTENTS, or CHECKURL-MULTI.

**CHECKURL-CONTENTS Size|UNKNOWN Filename** Indicates that the requested url has been verified to exist. The Size is the size in bytes, or use “UNKNOWN” if the size could not be determined. The Filename can be empty (in which case a default is used), or can specify a filename that is suggested to be used for this url.

**CHECKURL-MULTI Url Size|UNKNOWN Filename ...** Indicates that the requested url has been verified to exist, and contains multiple files, which can each be accessed using their own url. Note that since a list is returned, neither the Url nor the Filename can contain spaces.

**CHECKURL-FAILURE** Indicates that the requested url could not be accessed.

**req\_CLAIMURL** (*url*)

**req\_GETAVAILABILITY** ()

**req\_GETCOST** ()

**req\_INITREMOTE** (*\*args*)

Initialize this remote. Provides high level abstraction.

Specific implementation should go to `_initialize`

**req\_PREPARE** (*\*args*)

Prepare “to deliver”. Provides high level abstraction

Specific implementation should go to `_prepare`

**req\_REMOVE** (*key*)

**REMOVE-SUCCESS Key** Indicates the key has been removed from the remote. May be returned if the remote didn’t have the key at the point removal was requested.

**REMOVE-FAILURE Key ErrorMessage** Indicates that the key was unable to be removed from the remote.

**req\_TRANSFER** (*cmd, key, file*)

**req\_WHEREIS** (*key*)

Added in 5.20150812-17-g6bc46e3

provide any information about ways to access the content of a key stored in it, such as eg, public urls. This will be displayed to the user by eg, `git annex whereis`. The remote replies with WHEREIS-SUCCESS or WHEREIS-FAILURE. Note that users expect `git annex whereis` to run fast, without eg, network access.

This is not needed when SETURIPRESENT is used, since such uris are automatically displayed by git annex whereis.

**WHEREIS-SUCCESS String** Indicates a location of a key. Typically an url, the string can be anything that it makes sense to display to the user about content stored in the special remote.

**WHEREIS-FAILURE** Indicates that no location is known for a key.

**send** (\*args)

Send a message to git-annex

**Parameters** \*args (*list of strings*) – arguments to be joined by a space and passed to git-annex

**send\_unsupported** ()

**stop** (msg=None)

**class** datalad.customremotes.base.**AnnexExchangeProtocol** (repopath, custom\_remote\_name=None)

Bases: datalad.support.protocol.ProtocolInterface

A little helper to protocol interactions of custom remote with annex

**HEADER** = `#!/bin/bash\n\nset -e\n\n# Gets a VALUE response and stores it in $RET\nreport () {\n echo "$@" >&2\n}\n\n`

**add\_section** (cmd, exception)

**do\_execute\_callables**

**do\_execute\_ext\_commands**

**end\_section** (id\_, exception)

**initiate** ()

**records\_callables**

**records\_ext\_commands**

**start\_section** (cmd)

**write\_entries** (entries)

**write\_section** (cmd)

**exception** datalad.customremotes.base.**AnnexRemoteQuit**

Bases: exceptions.Exception

datalad.customremotes.base.**generate\_uuids** ()

Generate UUIDs for our remotes. Even though quick, for consistency pre-generated and recorded in consts.py

datalad.customremotes.base.**init\_datalad\_remote** (repo, remote, encryption=None, autoenable=False, opts=[])

Initialize datalad special remote

## datalad.customremotes.archives

Custom remote to support getting the load from archives present under annex

**class** datalad.customremotes.archives.**ArchiveAnnexCustomRemote** (persistent\_cache=True, \*\*kwargs)

Bases: *datalad.customremotes.base.AnnexCustomRemote*

Special custom remote allowing to obtain files from archives



Archives should also be under annex control.

**AVAILABILITY** = 'local'

**COST** = 500

**CUSTOM\_REMOTE\_NAME** = 'archive'

**SUPPORTED\_SCHEMES** = ('dl+archive',)

**URL\_PREFIX** = 'dl+archive:'

**URL\_SCHEME** = 'dl+archive'

**cache**

**get\_file\_url** (*archive\_file=None, archive\_key=None, file=None, size=None*)

Given archive (file or a key) and a file – compose URL for access

### Examples

**dl+archive:SHA256E-s176-69...3e.tar.gz#path=1/d2/2d&size=123** when size of file within archive was known to be 123

**dl+archive:SHA256E-s176-69...3e.tar.gz#path=1/d2/2d** when size of file within archive was not provided

**Parameters** **size** (*int, optional*) – Size of the file. If not provided, will simply be empty

**req\_CHECKPRESENT** (*key*)

Check if copy is available

TODO: just proxy the call to annex for underlying tarball

Replies

**CHECKPRESENT-SUCCESS Key** Indicates that a key has been positively verified to be present in the remote.

**CHECKPRESENT-FAILURE Key** Indicates that a key has been positively verified to not be present in the remote.

**CHECKPRESENT-UNKNOWN Key ErrorMessage** Indicates that it is not currently possible to verify if the key is present in the remote. (Perhaps the remote cannot be contacted.)

**req\_CHECKURL** (*url*)

Replies

**CHECKURL-CONTENTS Size|UNKNOWN Filename** Indicates that the requested url has been verified to exist. The Size is the size in bytes, or use "UNKNOWN" if the size could not be determined. The Filename can be empty (in which case a default is used), or can specify a filename that is suggested to be used for this url.

**CHECKURL-MULTI Url Size|UNKNOWN Filename ...** Indicates that the requested url has been verified to exist, and contains multiple files, which can each be accessed using their own url. Note that since a list is returned, neither the Url nor the Filename can contain spaces.

**CHECKURL-FAILURE** Indicates that the requested url could not be accessed.

**req\_REMOVE** (*key*)

**REMOVE-SUCCESS Key** Indicates the key has been removed from the remote. May be returned if the remote didn't have the key at the point removal was requested

**REMOVE-FAILURE Key ErrorMsg** Indicates that the key was unable to be removed from the remote.

**req\_WHEREIS** (*key*)

**WHEREIS-SUCCESS String** Indicates a location of a key. Typically an url, the string can be anything that it makes sense to display to the user about content stored in the special remote.

**WHEREIS-FAILURE** Indicates that no location is known for a key.

**stop** (*\*args*)

Stop communication with annex

`datalad.customremotes.archives.main()`  
cmdline entry point

## Configuration management

---

*config*

---

### datalad.config

**class** `datalad.config.ConfigManager` (*dataset=None, dataset\_only=False, overrides=None*)

Bases: `object`

Thin wrapper around *git-config* with support for a dataset configuration.

The general idea is to have an object that is primarily used to read/query configuration option. Upon creation, current configuration is read via one (or max two, in the case of the presence of dataset-specific configuration) calls to *git config*. If this class is initialized with a `Dataset` instance, it supports reading and writing configuration from `.datalad/config` inside a dataset too. This file is committed to Git and hence useful to ship certain configuration items with a dataset.

The API aims to provide the most significant read-access API of a dictionary, the Python `ConfigParser`, and `GitPython`'s config parser implementations.

This class is presently not capable of efficiently writing multiple configurations items at once. Instead, each modification results in a dedicated call to *git config*. This author thinks this is OK, as he cannot think of a situation where a large number of items need to be written during normal operation. If such need arises, various solutions are possible (via `GitPython`, or an independent writer).

Each instance carries a public *overrides* attribute. This dictionary contains variables that override any setting read from a file. The overrides are persistent across reloads, and are not modified by any of the manipulation methods, such as *set* or *unset*.

Any `DATALAD_*` environment variable is also presented as a configuration item. Settings read from environment variables are not stored in any of the configuration file, but are read dynamically from the environment at each *reload()* call. Their values take precedence over any specification in configuration files, and even overrides.

#### Parameters

- **dataset** (*Dataset, optional*) – If provided, all *git config* calls are executed in this dataset's directory. Moreover, any modifications are, by default, directed to this dataset's configuration file (which will be created on demand)
- **dataset\_only** (*bool*) – If True, configuration items are only read from a datasets persistent configuration file, if any present (the one in `.datalad/config`, not `.git/config`).

- **overrides** (*dict*, *optional*) – Variable overrides, see general class documentation for details.

**add** (*var*, *value*, *where*='dataset', *reload*=True)

Add a configuration variable and value

#### Parameters

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. 'core.editor'
- **value** (*str*) – Variable value
- **where** (*{'dataset', 'local', 'global'}*, *optional*) – Indicator which configuration file to modify. 'dataset' indicates the persistent configuration in *.datalad/config* of a dataset; 'local' the configuration of a dataset's Git repository in *.git/config*; 'global' refers to the general configuration that is not specific to a single repository (usually in *\$USER/.gitconfig*).
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

**get** (*k*, *d*) → D[k] if k in D, else d. d defaults to None.

**get\_value** (*section*, *option*, *default*=None)

Like *get()*, but with an optional default value

If the default is not None, the given default value will be returned in case the option did not exist. This behavior imitates GitPython's config parser.

**getbool** (*section*, *option*, *default*=None)

A convenience method which coerces the option value to a bool

Values "on", "yes", "true" and any int!=0 are considered True Values which evaluate to bool False, "off", "no", "false" are considered False TypeError is raised for other values.

**getfloat** (*section*, *option*)

A convenience method which coerces the option value to a float

**getint** (*section*, *option*)

A convenience method which coerces the option value to an integer

**has\_option** (*section*, *option*)

If the given section exists, and contains the given option

**has\_section** (*section*)

Indicates whether a section is present in the configuration

**items** (*section*=None)

Return a list of (name, value) pairs for each option

Optionally limited to a given section.

**keys** ()

Returns list of configuration item names

**obtain** (*var*, *default*=None, *dialog\_type*=None, *valtype*=None, *store*=False, *where*=None, *reload*=True, *\*\*kwargs*)

Convenience method to obtain settings interactively, if needed

A UI will be used to ask for user input in interactive sessions. Questions to ask, and additional explanations can be passed directly as arguments, or retrieved from a list of pre-configured items.

Additionally, this method allows for type conversion and storage of obtained settings. Both aspects can also be pre-configured.

### Parameters

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. ‘core.editor’
- **default** (*any type*) – In interactive sessions and if *store* is True, this default value will be presented to the user for confirmation (or modification). In all other cases, this value will be silently assigned unless there is an existing configuration setting.
- **dialog\_type** (*{'question', 'yesno', None}*) – Which dialog type to use in interactive sessions. If *None*, pre-configured UI options are used.
- **store** (*bool*) – Whether to store the obtained value (or default)
- **where** (*{'dataset', 'local', 'global'}, optional*) – Indicator which configuration file to modify. ‘dataset’ indicates the persistent configuration in *.datalad/config* of a dataset; ‘local’ the configuration of a dataset’s Git repository in *.git/config*; ‘global’ refers to the general configuration that is not specific to a single repository (usually in *\$USER/.gitconfig*).
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.
- **\*\*kwargs** – Additional arguments for the UI function call, such as a question *text*.

### options (*section*)

Returns a list of options available in the specified section.

### reload (*force=False*)

Reload all configuration items from the configured sources

If *force* is False, all files configuration was previously read from are checked for differences in the modification times. If no difference is found for any file no reload is performed. This mechanism will not detect newly created global configuration files, use *force* in this case.

### remove\_section (*sec, where='dataset', reload=True*)

Rename a configuration section

### Parameters

- **sec** (*str*) – Name of the section to remove.
- **where** (*{'dataset', 'local', 'global'}, optional*) – Indicator which configuration file to modify. ‘dataset’ indicates the persistent configuration in *.datalad/config* of a dataset; ‘local’ the configuration of a dataset’s Git repository in *.git/config*; ‘global’ refers to the general configuration that is not specific to a single repository (usually in *\$USER/.gitconfig*).
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

### rename\_section (*old, new, where='dataset', reload=True*)

Rename a configuration section

### Parameters

- **old** (*str*) – Name of the section to rename.
- **new** (*str*) – Name of the section to rename to.
- **where** (*{'dataset', 'local', 'global'}, optional*) – Indicator which configuration file to modify. ‘dataset’ indicates the persistent configuration in *.datalad/config* of a dataset; ‘local’ the configuration of a dataset’s Git repository in *.git/config*;

‘global’ refers to the general configuration that is not specific to a single repository (usually in \$USER/.gitconfig).

- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

#### **sections** ()

Returns a list of the sections available

**set** (*var*, *value*, *where*=‘dataset’, *reload*=True, *force*=False)

Set a variable to a value.

In opposition to *add*, this replaces the value of *var* if there is one already.

#### **Parameters**

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. ‘core.editor’
- **value** (*str*) – Variable value
- **force** (*bool*) – if set, replaces all occurrences of *var* by a single one with the given *value*. Otherwise raise if multiple entries for *var* exist already
- **where** ({‘dataset’, ‘local’, ‘global’}, *optional*) – Indicator which configuration file to modify. ‘dataset’ indicates the persistent configuration in .datalad/config of a dataset; ‘local’ the configuration of a dataset’s Git repository in .git/config; ‘global’ refers to the general configuration that is not specific to a single repository (usually in \$USER/.gitconfig).
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

**unset** (*var*, *where*=‘dataset’, *reload*=True)

Remove all occurrences of a variable

#### **Parameters**

- **var** (*str*) – Name of the variable to remove
- **where** ({‘dataset’, ‘local’, ‘global’}, *optional*) – Indicator which configuration file to modify. ‘dataset’ indicates the persistent configuration in .datalad/config of a dataset; ‘local’ the configuration of a dataset’s Git repository in .git/config; ‘global’ refers to the general configuration that is not specific to a single repository (usually in \$USER/.gitconfig).
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

datalad.config.**anything2bool** (*val*)

datalad.config.**get\_git\_version** (*runner*)

Return version of available git

## Crawler

<i>crawler.base</i>	Crawling of external resources (e.g.
<i>crawler.pipeline</i>	Pipeline functionality.

## **datalad.crawler.base**

Crawling of external resources (e.g. websites) to get/update datasets

## **datalad.crawler.pipeline**

Pipeline functionality.

A pipeline is represented by a simple list or tuple of nodes or other nested pipelines. Each pipeline node is a callable which receives a dictionary (commonly named *data*), does some processing, and yields (once or multiple times) a derived dictionary (commonly a shallow copy of original dict). For a node to be parametrized it should be implemented as a callable (i.e. define `__call__`) class, which could obtain parameters in its constructor.

TODO: describe PIPELINE\_OPTS and how to specify them for a given (sub-)pipeline.

The *data* dictionary is used primarily to carry the scraped/produced data, but besides that it will carry few items which some nodes might use. All those item names will start with the *datalad\_* prefix, and will be intended for 'inplace' modifications or querying. The following items are planned to be provided by the pipeline runner:

***datalad\_settings*** PipelineSettings object which could be used to provide configuration for the current run of the pipeline. E.g.:

- *dry*: either nodes are intended not to perform any changes which would reflect on disk
- *skip\_existing*:

***datalad\_stats*** ActivityStats/dict object to accumulate statistics on what has been done by the nodes so far

To some degree, we could make an analogy when *blood* is to *data* and *venous system* is to *pipeline*. Blood delivers various elements which are picked up by various parts of our body when they know what to do with the corresponding elements. To the same degree nodes can consume, augment, or produce new items to the *data* and send it down the stream. Since there is no strict typing or specification on what nodes could consume or produce (yet), no verification is done and things can go utterly wrong. So nodes must be robust and provide informative logging.

### **exception datalad.crawler.pipeline.FinishPipeline**

Bases: `exceptions.Exception`

Exception to use to signal that any given pipeline should be stopped

`datalad.crawler.pipeline.get_repo_pipeline_config_path(repo_path='.')`

Given a path within a repo, return path to the `crawl.cfg`

`datalad.crawler.pipeline.get_repo_pipeline_script_path(repo_path='.')`

If there is a single pipeline present among 'pipelines/', return path to it

`datalad.crawler.pipeline.initiate_pipeline_config(template, template_func=None, template_kwargs=None, path='.', commit=False)`

TODO Gergana ;)

`datalad.crawler.pipeline.load_pipeline_from_config(path)`

Given a path to the pipeline configuration file, instantiate a pipeline

Typical example description

```
[crawl:pipeline] pipeline = standard func = pipeline1 _kward1 = 1
```

which would instantiate a pipeline from `standard.py` module by calling `standard.pipeline1` with `_kward1='1'`. This definition is identical to

```
[crawl:pipeline] pipeline = standard?func=pipeline1&_kward1=1
```

so that theoretically we could specify basic pipelines completely within a URL

`datalad.crawler.pipeline.load_pipeline_from_module` (*module*, *func=None*,  
*args=None*, *kwargs=None*,  
*return\_only=False*)

Load pipeline from a Python module

**Parameters**

- **module** (*str*) – Module name or filename of the module from which to load the pipeline
- **func** (*str*, *optional*) – Function within the module to use. Default: *pipeline*
- **args** (*list or tuple*, *optional*) – Positional arguments to provide to the function.
- **kwargs** (*dict*, *optional*) – Keyword arguments to provide to the function.
- **return\_only** (*bool*, *optional*) – flag true if only to return pipeline

`datalad.crawler.pipeline.load_pipeline_from_template` (*name*, *func=None*,  
*args=None*, *kwargs=None*,  
*return\_only=False*)

Given a name, loads that pipeline from `datalad.crawler.pipelines`  
and later from other locations

**Parameters**

- **name** (*str*) – Name of the pipeline (the template) defining the filename, or the full path to it (TODO), example: `openfmri`
- **func** (*str*) – Name of function from which pipeline to run example: `superdataset_pipeline`
- **args** (*dict*, *optional*) – Positional args for the pipeline, passed as *\*args* into the pipeline call
- **kwargs** (*dict*, *optional*) – Keyword args for the pipeline, passed as *\*\*kwargs* into the pipeline call, example: `{‘dataset’: ‘ds000001’}`
- **return\_only** (*bool*, *optional*) – flag true if only to return pipeline

`datalad.crawler.pipeline.reset_pipeline` (*pipeline*)

Given a pipeline, traverse its nodes and call `.reset` on them

Note: it doesn’t try to call `reset` if a node doesn’t have it

`datalad.crawler.pipeline.run_pipeline` (*\*args*, *\*\*kwargs*)

Run pipeline and assemble results into a list

By default, the pipeline returns only its input (see `PIPELINE_OPTS`), so if no options for the pipeline were given to return additional items, a `[[{}]]` will be provided as output

`datalad.crawler.pipeline.xrun_pipeline` (*pipeline*, *data=None*, *stats=None*, *reset=True*)

Yield results from the pipeline.

`datalad.crawler.pipeline.xrun_pipeline_steps` (*pipeline*, *data*, *output=‘input’*)

Actually run pipeline steps, feeding yielded results to the next node and yielding results back.

Recursive beast which runs a single node and then recurses to run the rest, possibly multiple times if the current node is a generator. It yields output from the node/nested pipelines, as directed by the output argument.

**Test infrastructure**

Continued on next page
------------------------

Table 5.8 – continued from previous page

<code>tests.utils</code>	Miscellaneous utilities to assist with testing
<code>tests.utils_testrepos</code>	
<code>tests.heavyoutput</code>	Helper to provide heavy load on stdout and stderr

## datalad.tests.utils

Miscellaneous utilities to assist with testing

**class** `datalad.tests.utils.SilentHTTPHandler` (*\*args*, *\*\*kwargs*)

Bases: `SimpleHTTPServer.SimpleHTTPRequestHandler`

A little adapter to silence the handler

**log\_message** (*format*, *\*args*)

`datalad.tests.utils.assert_dict_equal` (*d1*, *d2*)

`datalad.tests.utils.assert_in_results` (*results*, *\*\*kwargs*)

Verify that the particular combination of keys and values is found in one of the results

`datalad.tests.utils.assert_is_generator` (*gen*)

`datalad.tests.utils.assert_message` (*message*, *results*)

Verify that each status dict in the results has a message

This only tests the message template string, and not a formatted message with args expanded.

`datalad.tests.utils.assert_no_errors_logged` (*func*, *skip\_re=None*)

Decorator around function to assert that no errors logged during its execution

`datalad.tests.utils.assert_not_in_results` (*results*, *\*\*kwargs*)

Verify that the particular combination of keys and values is not in any of the results

`datalad.tests.utils.assert_re_in` (*regex*, *c*, *flags=0*, *match=True*, *msg=None*)

Assert that container (list, str, etc) contains entry matching the regex

`datalad.tests.utils.assert_result_count` (*results*, *n*, *\*\*kwargs*)

Verify specific number of results (matching criteria, if any)

`datalad.tests.utils.assert_result_values_equal` (*results*, *prop*, *values*)

Verify that the values of all results for a given key in the status dicts match the given sequence

`datalad.tests.utils.assert_status` (*label*, *results*)

Verify that each status dict in the results has a given status label

*label* can be a sequence, in which case status must be one of the items in this sequence.

`datalad.tests.utils.clone_url` (*url*)

`datalad.tests.utils.create_tree` (*path*, *tree*, *archives\_leading\_dir=True*)

Given a list of tuples (name, load) create such a tree

if load is a tuple itself – that would create either a subtree or an archive with that content and place it into the tree if name ends with `.tar.gz`

`datalad.tests.utils.create_tree_archive` (*path*, *name*, *load*, *overwrite=False*, *archives\_leading\_dir=True*)

Given an archive *name*, create under *path* with specified *load* tree

`datalad.tests.utils.dump_graph` (*graph*, *flatten=False*)



`datalad.tests.utils.get_most_obscure_supported_name(*arg, **kw)`  
 Return the most obscure filename that the filesystem would support under TEMPDIR

TODO: we might want to use it as a function where we would provide tdir

`datalad.tests.utils.get_mtimes_and_digests(target_path)`  
 Return digests (md5) and mtimes for all the files under target\_path

`datalad.tests.utils.ignore_nose_capturing_stdout(func)`  
 Decorator workaround for nose's behaviour with redirecting sys.stdout

Needed for tests involving the runner and nose redirecting stdout. Counter-intuitively, that means it needed for nosetests without '-s'. See issue reported here: <https://code.google.com/p/python-nose/issues/detail?id=243&can=1&sort=-id&colspec=ID%20Type%20Status%20Priority%20Stars%20Milestone%20Owner%20Summary>

`datalad.tests.utils.integration(f)`  
 Mark test as an "integration" test which generally is not needed to be run

Generally tend to be slower

`datalad.tests.utils.nok_startswith(s, prefix)`

`datalad.tests.utils.ok_annex_get(ar, files, network=True)`  
 Helper to run .get decorated checking for correct operation

get passes through stderr from the ar to the user, which pollutes screen while running tests

Note: Currently not true anymore, since usage of -json disables progressbars

`datalad.tests.utils.ok_archives_caches(repopath, n=1, persistent=None)`  
 Given a path to repository verify number of archives

#### Parameters

- **repopath** (*str*) – Path to the repository
- **n** (*int, optional*) – Number of archives directories to expect
- **persistent** (*bool or None, optional*) – If None – both persistent and not count.

`datalad.tests.utils.ok_broken_symlink(path)`

`datalad.tests.utils.ok_clean_git(path, annex=None, head_modified=[], index_modified=[], untracked=[], ignore_submodules=False)`

Verify that under given path there is a clean git repository

it exists, .git exists, nothing is uncommitted/dirty/staged

---

**Note:** Parameters head\_modified and index\_modified currently work in pure git or indirect mode annex only and are ignored otherwise! Implementation is yet to do!

---

#### Parameters

- **path** (*str or Repo*) – in case of a str: path to the repository's base dir; Note, that passing a Repo instance prevents detecting annex. This might be useful in case of a non-initialized annex, a GitRepo is pointing to.
- **annex** (*bool or None*) – explicitly set to True or False to indicate, that an annex is (not) expected; set to None to autodetect, whether there is an annex. Default: None.
- **ignore\_submodules** (*bool*) – if True, submodules are not inspected

`datalad.tests.utils.ok_endswith` (*s*, *suffix*)

`datalad.tests.utils.ok_exists` (*path*)

`datalad.tests.utils.ok_file_has_content` (*path*, *content*, *strip=False*, *re=False*, *\*\*kwargs*)  
Verify that file exists and has expected content

`datalad.tests.utils.ok_file_under_git` (*path*, *filename=None*, *annexed=False*)  
Test if file is present and under git/annex control

If relative path provided, then test from current directory

`datalad.tests.utils.ok_generator` (*gen*)

`datalad.tests.utils.ok_git_config_not_empty` (*ar*)  
Helper to verify that nothing rewritten the config file

`datalad.tests.utils.ok_good_symlink` (*path*)

`datalad.tests.utils.ok_startswith` (*s*, *prefix*)

`datalad.tests.utils.ok_symlink` (*path*)  
Checks whether path is either a working or broken symlink

`datalad.tests.utils.put_file_under_git` (*path*, *filename=None*, *content=None*, *annexed=False*)  
Place file under git/annex and return used Repo

`datalad.tests.utils.skip_httppretty_on_problematic_pythons` (*func*)

As discovered some httppretty bug causes a side-effect on other tests on some Pythons. So we skip the test if such problematic combination detected

References <https://travis-ci.org/datalad/datalad/jobs/94464988> <http://stackoverflow.com/a/29603206/1265472>

`datalad.tests.utils.skip_if_no_module` (*module*)

`datalad.tests.utils.skip_if_no_network` (*func=None*)  
Skip test completely in NONNETWORK settings

If not used as a decorator, and just a function, could be used at the module level

`datalad.tests.utils.skip_if_on_windows` (*func*)

Skip test completely under Windows

`datalad.tests.utils.skip_if_scrapy_without_selector` ()

A little helper to skip some tests which require recent scrapy

`datalad.tests.utils.skip_if_url_is_not_available` (*url*, *regex=None*)

`datalad.tests.utils.skip_ssh` (*func*)

Skips SSH tests if on windows or if environment variable DATALAD\_TESTS\_SSH was not set

`datalad.tests.utils.slow` (*f*)

Mark test as a slow, although not necessarily integration or usecase test

`datalad.tests.utils.usecase` (*f*)

Mark test as a usecase user ran into and which (typically) caused bug report to be filed/troubleshooted

### **datalad.tests.utils\_testrepos**

**class** `datalad.tests.utils_testrepos.BasicAnnexTestRepo` (*path=None*,  
*puke\_if\_exists=True*)

Bases: `datalad.tests.utils_testrepos.TestRepo`

Creates a basic test git-annex repository

**REPO\_CLASS**  
alias of AnnexRepo

**create\_info\_file()**

**populate()**

**class** datalad.tests.utils\_testrepos.**BasicGitTestRepo** (*path=None, puke\_if\_exists=True*)  
Bases: *datalad.tests.utils\_testrepos.TestRepo*

Creates a basic test git repository.

**REPO\_CLASS**  
alias of GitRepo

**create\_info\_file()**

**populate()**

**class** datalad.tests.utils\_testrepos.**InnerSubmodule**  
Bases: *object*

**create()**

**path**

**url**

**class** datalad.tests.utils\_testrepos.**NestedDataset** (*path=None, puke\_if\_exists=True*)  
Bases: *datalad.tests.utils\_testrepos.BasicAnnexTestRepo*

**populate()**

**class** datalad.tests.utils\_testrepos.**SubmoduleDataset** (*path=None, puke\_if\_exists=True*)  
Bases: *datalad.tests.utils\_testrepos.BasicAnnexTestRepo*

**populate()**

**class** datalad.tests.utils\_testrepos.**TestRepo** (*path=None, puke\_if\_exists=True*)  
Bases: *object*

**REPO\_CLASS = None**

**create()**

**create\_file** (*name, content, add=True, annex=False*)

**path**

**populate()**

**url**

## **datalad.tests.heavyoutput**

Helper to provide heavy load on stdout and stderr

## **Command line interface infrastructure**

---

*cmdline.main*

---

*cmdline.helpers*

---

*cmdline.common\_args*

---

## datalad.cmdline.main

datalad.cmdline.main.**main** (*args=None*)

datalad.cmdline.main.**setup\_parser** (*formatter\_class=<class parse.RawDescriptionHelpFormatter>*, *arg-  
return\_subparsers=False*)

## datalad.cmdline.helpers

**class** datalad.cmdline.helpers.**HelpAction** (*option\_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

**class** datalad.cmdline.helpers.**LogLevelAction** (*option\_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None*)

Bases: `argparse.Action`

**class** datalad.cmdline.helpers.**RegexType**

Bases: `object`

Factory for creating regular expression types for argparse

DEPRECATED AFAIK – now things are in the config file, but we might provide a mode where we operate solely from cmdline

datalad.cmdline.helpers.**get\_repo\_instance** (*path='.', class\_=None*)

Returns an instance of appropriate datalad repository for path. Check whether a certain path is inside a known type of repository and returns an instance representing it. May also check for a certain type instead of detecting the type of repository.

### Parameters

- **path** (*str*) – path to check; default: current working directory
- **class** (*class*) – if given, check whether path is inside a repository, that can be represented as an instance of the passed class.

**Raises** `RuntimeError`, in case cwd is not inside a known repository.

datalad.cmdline.helpers.**parser\_add\_common\_opt** (*parser, opt, names=None, \*\*kwargs*)

datalad.cmdline.helpers.**run\_via\_pbs** (*args, pbs*)

datalad.cmdline.helpers.**strip\_arg\_from\_argv** (*args, value, opt\_names*)

Strip an originally listed option (with its value) from the list cmdline args

## datalad.cmdline.common\_args

## Configuration

Datalad uses the same configuration mechanism and syntax as Git itself. Consequently, datalad can be configured using the `git config` command. Both a *global* user configuration (typically at `~/.gitconfig`), and a *local* repository-specific configuration (`.git/config`) are inspected.

In addition, datalad supports a persistent dataset-specific configuration. This configuration is stored at `.datalad/config` in any dataset. As it is part of a dataset, settings stored there will also be in effect for any consumer of such a dataset. Both *global* and *local* settings on a particular machine always override configuration shipped with a dataset.

All datalad-specific configuration variables are prefixed with `datalad.`.

It is possible to override or amend the configuration using environment variables. Any variable with a name that starts with `DATALAD_` will be available as the corresponding `datalad.` configuration variable, replacing any `_` in the name with a dot, and all letters converted to lower case. Values from environment variables take precedence over configuration file settings.

The following sections provide a (non-exhaustive) list of settings honored by datalad. They are categorized according to the scope they are typically associated with.

## Global user configuration

**datalad.crawl.init\_direct** Default annex repository mode: Should dataset be initialized in direct mode?

**datalad.crawl.pipeline.housekeeping** Crawler pipeline house keeping: Should the crawler tidy up datasets (git gc, repack, clean)?

[value must be convertible to type bool]

**datalad.externals.nda.dbserver** NDA database server: Hostname of the database server

**datalad.locations.cache** Cache directory: Where should datalad cache files? Default: `~/cache/datalad`

## Local repository configuration

**datalad.crawl.cache** Crawler download caching: Should the crawler cache downloaded files?

[bool]

**datalad.crawl.dryrun** Crawler dry-run: Should the crawler ... I AM NOT QUITE SURE WHAT?

[value must be convertible to type bool]

## Sticky dataset configuration

**datalad.crawl.default\_backend** Default annex backend: Content hashing method to be used by git-annex

## Miscellaneous configuration

**datalad.cmd.protocol** Specifies the protocol number used by the Runner to note shell command or python function call times and allows for dry runs. “externals-time” for ExecutionTimeExternalsProtocol, “time” for ExecutionTimeProtocol and “null” for NullProtocol. Any new `DATALAD_CMD_PROTOCOL` has to implement `datalad.support.protocol.ProtocolInterface`:

**datalad.cmd.protocol.prefix** Sets a prefix to add before the command call times are noted by `DATALAD_CMD_PROTOCOL`.

**datalad.exc.str.tblimit** This flag is used by the `datalad extract_tb` function which extracts and formats stack-traces. It caps the number of lines to `DATALAD_EXC_STR_TBLIMIT` of pre-processed entries from traceback.

**datalad.log.level** Used for control the verbosity of logs printed to stdout while running datalad commands/debugging:

**datalad.log.name** Include name of the log target in the log line:

**datalad.log.names** Which names (,-separated) to print log lines for:

**datalad.log.namesre** Regular expression for which names to print log lines for:

**datalad.log.outputs** Used to control either both stdout and stderr of external commands execution are logged in detail (at DEBUG level):

**datalad.log.timestamp** Used to add timestamp to datalad logs: Default: False

[value must be convertible to type bool]

**datalad.log.traceback** Runs TraceBack function with collide set to True, if this flag is set to “collide”. This replaces any common prefix between current traceback log and previous invocation with “...”:

**datalad.repo.direct** Direct Mode for git-annex repositories: Set this flag to create annex repositories in direct mode by default

[value must be convertible to type bool]

**datalad.repo.version** git-annex repository version: Specifies the repository version for git-annex to be used by default

[value must be convertible to type ‘int’]

**datalad.tests.dataladremote** Binary flag to specify whether each annex repository should get datalad special remote in every test repository:

[value must be convertible to type bool]

**datalad.tests.nonetwork** Skips network tests completely if this flag is set Examples include test for s3, git\_repositories, openfmri etc:

[value must be convertible to type bool]

**datalad.tests.nonlo** Specifies network interfaces to bring down/up for testing. Currently used by travis.:

**datalad.tests.noteardown** Does not execute teardown\_package which cleans up temp files and directories created by tests if this flag is set:

[value must be convertible to type bool]

**datalad.tests.protocolremote** Binary flag to specify whether to test protocol interactions of custom remote with annex:

[value must be convertible to type bool]

**datalad.tests.runcmdline** Binary flag to specify if shell testing using shunit2 to be carried out:

[value must be convertible to type bool]

**datalad.tests.ssh** Skips SSH tests if this flag is **not** set:

[value must be convertible to type bool]

**datalad.tests.temp.dir** Create a temporary directory at location specified by this flag. It is used by tests to create a temporary git directory while testing git annex archives etc:

**datalad.tests.temp.fs** Specify the temporary file system to use as loop device for testing DATA-LAD\_TESTS\_TEMP\_DIR creation:

**datalad.tests.temp.fssize** Specify the size of temporary file system to use as loop device for testing DATA-LAD\_TESTS\_TEMP\_DIR creation:

**datalad.tests.temp.keep** Function rmtmp will not remove temporary file/directory created for testing if this flag is set:

[value must be convertible to type bool]

**datalad.tests.ui.backend** Tests UI backend: Which UI backend to use Default: tests-noninteractive

**datalad.tests.usecassette** Specifies the location of the file to record network transactions by the VCR module. Currently used by when testing custom special remotes:





## Automatic creation and maintenance of datasets by crawling external resources

### DataLad Crawler 101

#### Nodes

A node in a pipeline is just a callable (function or a method of a class) which takes a dictionary, and yields a dictionary any number of times. The simplest node could look like

```
>>> def add1_node(data, field='input'):
...     data_ = data.copy()
...     data_['input'] += 1
...     yield data_
```

which creates a simple node, intended to increment an arbitrary (specified by *field* keyword argument) field in the input dictionary and yields a modified dictionary as its output once.

```
>>> next(add1_node({'input': 1}))
{'input': 2}
```

Nodes are generators which yield a dictionary zero, one, or multiple times and yield a dictionary. For more on generators, reference the Python documentation on [Generators](#).

---

**Note:** Nodes should not have side-effects, i.e. they should not modify input data, but yield a shallow copy if any of the field values need to be added, removed, or modified. To help with creation of a new shallow copy with some fields adjusted, use `updated()`.

---

## Pipelines

A pipeline is a series of generators ordered into a list. Each generator takes the output of its predecessor as its own input. The first node in the pipeline would need to be provided with specific input. The simplest pipeline could look like

```
>>> from datalad.crawler.nodes.crawl_url import crawl_url
>>> from datalad.crawler.nodes.matches import a_href_match
>>> from datalad.crawler.nodes.annex import Annexificator
>>> annex = Annexificator(allow_dirty=True) # so we could demo right within
>>> pipeline = \
...     [
...         crawl_url('http://map.org/datasets'),
...         a_href_match(".*\.mat"),
...         annex
...     ]
```

in which the first node (method of a class) is provided with input and crawls a website. *a\_href\_match* then works to output all files that end in *.mat*, and those files are lastly inputted to *annex*, another node, which simply annexes them.

---

**Note:** Since pipelines depend heavily on nodes, these nodes must yield in order for an output to be produced. If a generator fails to yield, then the pipeline can no longer continue and it is stopped at that node.

---

## Subpipelines

A subpipeline is a pipeline that lives within a greater pipeline and is also denoted by *[]*. Two subpipelines that exist on top of one another will take in the same input, but process it with different generators. This functionality allows for the same input to be handled in two or more (depending on the number of subpipelines) different manners.

TODO: 'FinishPipeline' exception here *FinishPipeline*

## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



## d

- [datalad.auto](#), 131
- [datalad.cmd](#), 132
- [datalad.cmdline.common\\_args](#), 168
- [datalad.cmdline.helpers](#), 168
- [datalad.cmdline.main](#), 168
- [datalad.config](#), 158
- [datalad.consts](#), 134
- [datalad.crawler.base](#), 162
- [datalad.crawler.pipeline](#), 162
- [datalad.customremotes.archives](#), 156
- [datalad.customremotes.base](#), 154
- [datalad.customremotes.main](#), 154
- [datalad.log](#), 134
- [datalad.support.annexrepo](#), 139
- [datalad.support.archives](#), 151
- [datalad.support.configparserinc](#), 153
- [datalad.tests.heavyoutput](#), 167
- [datalad.tests.utils](#), 164
- [datalad.tests.utils\\_testrepos](#), 166
- [datalad.utils](#), 134
- [datalad.version](#), 139



## A

- activate() (datalad.auto.AutomagicIO method), 132
- active (datalad.auto.AutomagicIO attribute), 132
- add() (datalad.config.ConfigManager method), 159
- add() (datalad.support.annexrepo.AnnexRepo method), 140
- add() (in module datalad.api), 94
- add\_archive\_content() (in module datalad.api), 129
- add\_remote() (datalad.support.annexrepo.AnnexRepo method), 140
- add\_section() (datalad.customremotes.base.AnnexExchangeProtocol method), 156
- add\_url\_to\_file() (datalad.support.annexrepo.AnnexRepo method), 140
- add\_urls() (datalad.support.annexrepo.AnnexRepo method), 140
- adjust() (datalad.support.annexrepo.AnnexRepo method), 141
- aggregate\_metadata() (in module datalad.api), 115
- annex, 28
- AnnexCustomRemote (class in datalad.customremotes.base), 154
- AnnexExchangeProtocol (class in datalad.customremotes.base), 156
- AnnexRemoteQuit, 156
- AnnexRepo (class in datalad.support.annexrepo), 139
- annotate\_paths() (in module datalad.api), 116
- any\_re\_search() (in module datalad.utils), 134
- anything2bool() (in module datalad.config), 161
- ArchiveAnnexCustomRemote (class in datalad.customremotes.archives), 156
- ArchivesCache (class in datalad.support.archives), 151
- assert\_dict\_equal() (in module datalad.tests.utils), 164
- assert\_in\_results() (in module datalad.tests.utils), 164
- assert\_is\_generator() (in module datalad.tests.utils), 164
- assert\_message() (in module datalad.tests.utils), 164
- assert\_no\_errors\_logged() (in module datalad.tests.utils), 164
- assert\_not\_in\_results() (in module datalad.tests.utils), 164
- assert\_re\_in() (in module datalad.tests.utils), 164
- assert\_result\_count() (in module datalad.tests.utils), 164
- assert\_result\_values\_equal() (in module datalad.tests.utils), 164
- assert\_status() (in module datalad.tests.utils), 164
- assure\_bool() (in module datalad.utils), 134
- assure\_dict\_from\_str() (in module datalad.utils), 134
- assure\_dir() (in module datalad.utils), 134
- assure\_extracted() (datalad.support.archives.ExtractedArchive method), 152
- assure\_list() (in module datalad.utils), 134
- assure\_list\_from\_str() (in module datalad.utils), 134
- assure\_tuple\_or\_list() (in module datalad.utils), 134
- assure\_unicode() (in module datalad.utils), 134
- auto\_repr() (in module datalad.utils), 134
- autoget (datalad.auto.AutomagicIO attribute), 132
- AutomagicIO (class in datalad.auto), 131
- AVAILABILITY (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157
- AVAILABILITY (datalad.customremotes.base.AnnexCustomRemote attribute), 154

## B

- BasicAnnexTestRepo (class in datalad.tests.utils\_testrepos), 166
- BasicGitTestRepo (class in datalad.tests.utils\_testrepos), 167
- BatchedAnnex (class in datalad.support.annexrepo), 151
- BatchedAnnexes (class in datalad.support.annexrepo), 151
- better\_wraps() (in module datalad.utils), 135

## C

- cache (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157
- call() (datalad.cmd.Runner method), 132

chpwd (class in datalad.utils), 135  
 clean() (datalad.support.archives.ArchivesCache method), 152  
 clean() (datalad.support.archives.ExtractedArchive method), 152  
 clean() (in module datalad.api), 118  
 clear() (datalad.support.annexrepo.BatchedAnnexes method), 151  
 clone() (in module datalad.api), 119  
 clone\_url() (in module datalad.tests.utils), 164  
 close() (datalad.support.annexrepo.BatchedAnnex method), 151  
 close() (datalad.support.annexrepo.BatchedAnnexes method), 151  
 ColorFormatter (class in datalad.log), 134  
 commands (datalad.cmd.Runner attribute), 132  
 commit() (datalad.support.annexrepo.AnnexRepo method), 141  
 compress\_files() (in module datalad.support.archives), 152  
 ConfigManager (class in datalad.config), 158  
 copy\_to() (datalad.support.annexrepo.AnnexRepo method), 141  
 COST (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157  
 COST (datalad.customremotes.base.AnnexCustomRemote attribute), 154  
 crawl() (in module datalad.api), 130  
 crawl\_init() (in module datalad.api), 131  
 create() (datalad.tests.utils\_testrepos.InnerSubmodule method), 167  
 create() (datalad.tests.utils\_testrepos.TestRepo method), 167  
 create() (in module datalad.api), 96  
 create\_file() (datalad.tests.utils\_testrepos.TestRepo method), 167  
 create\_info\_file() (datalad.tests.utils\_testrepos.BasicAnnexTestRepo method), 167  
 create\_info\_file() (datalad.tests.utils\_testrepos.BasicGitTestRepo method), 167  
 create\_sibling() (in module datalad.api), 98  
 create\_sibling\_github() (in module datalad.api), 100  
 create\_test\_dataset() (in module datalad.api), 121  
 create\_tree() (in module datalad.tests.utils), 164  
 create\_tree\_archive() (in module datalad.tests.utils), 164  
 CUSTOM\_REMOTE\_NAME (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157  
 CUSTOM\_REMOTE\_NAME (datalad.customremotes.base.AnnexCustomRemote attribute), 154  
 cwd (datalad.cmd.Runner attribute), 132

## D

datalad.auto (module), 131  
 datalad.cmd (module), 132  
 datalad.cmd.protocol, 169  
 datalad.cmd.protocol.prefix, 169  
 datalad.cmdline.common\_args (module), 168  
 datalad.cmdline.helpers (module), 168  
 datalad.cmdline.main (module), 168  
 datalad.config (module), 158  
 datalad.consts (module), 134  
 datalad.crawl.cache, 169  
 datalad.crawl.default\_backend, 169  
 datalad.crawl.dryrun, 169  
 datalad.crawl.init\_direct, 169  
 datalad.crawl.pipeline.housekeeping, 169  
 datalad.crawler.base (module), 162  
 datalad.crawler.pipeline (module), 162  
 datalad.customremotes.archives (module), 156  
 datalad.customremotes.base (module), 154  
 datalad.customremotes.main (module), 154  
 datalad.exc.str.tblimit, 169  
 datalad.externals.nda.dbserver, 169  
 datalad.locations.cache, 169  
 datalad.log (module), 134  
 datalad.log.level, 169  
 datalad.log.name, 169  
 datalad.log.names, 170  
 datalad.log.namesre, 170  
 datalad.log.outputs, 170  
 datalad.log.timestamp, 170  
 datalad.log.traceback, 170  
 datalad.repo.direct, 170  
 datalad.repo.version, 170  
 datalad.support.annexrepo (module), 139  
 datalad.support.archives (module), 151  
 datalad.support.configparserinc (module), 153  
 datalad.tests.dataladremote, 170  
 datalad.tests.heavyoutput (module), 167  
 datalad.tests.nonnetwork, 170  
 datalad.tests.nonlo, 170  
 datalad.tests.noteardown, 170  
 datalad.tests.protocolremote, 170  
 datalad.tests.runcmdline, 170  
 datalad.tests.ssh, 170  
 datalad.tests.temp.dir, 170  
 datalad.tests.temp.fs, 170  
 datalad.tests.temp.fssize, 170  
 datalad.tests.temp.keep, 170  
 datalad.tests.ui.backend, 171  
 datalad.tests.usecassette, 171  
 datalad.tests.utils (module), 164  
 datalad.tests.utils\_testrepos (module), 166  
 datalad.utils (module), 134  
 datalad.version (module), 139



- dataset, 28
- deactivate() (datalad.auto.AutomagicIO method), 132
- debug() (datalad.customremotes.base.AnnexCustomRemote method), 154
- decompress\_file() (in module datalad.support.archives), 153
- default\_backends (datalad.support.annexrepo.AnnexRepo attribute), 141
- diff() (in module datalad.api), 121
- do\_execute\_callables (datalad.customremotes.base.AnnexExchangeProtocol attribute), 156
- do\_execute\_ext\_commands (datalad.customremotes.base.AnnexExchangeProtocol attribute), 156
- download\_url() (in module datalad.api), 123
- drop() (datalad.support.annexrepo.AnnexRepo method), 141
- drop() (in module datalad.api), 101
- drop\_key() (datalad.support.annexrepo.AnnexRepo method), 142
- dry (datalad.cmd.Runner attribute), 132
- dump\_graph() (in module datalad.tests.utils), 164
- ## E
- enable\_remote() (datalad.support.annexrepo.AnnexRepo method), 142
- encode\_filename() (in module datalad.utils), 135
- end\_section() (datalad.customremotes.base.AnnexExchangeProtocol method), 156
- env (datalad.cmd.Runner attribute), 132
- environment variable  
PATH, 28
- error() (datalad.customremotes.base.AnnexCustomRemote method), 154
- escape\_filename() (in module datalad.utils), 135
- expandpath() (in module datalad.utils), 135
- export() (in module datalad.api), 103
- ExtractedArchive (class in datalad.support.archives), 152
- ## F
- file\_basename() (in module datalad.utils), 135
- file\_has\_content() (datalad.support.annexrepo.AnnexRepo method), 142
- find() (datalad.support.annexrepo.AnnexRepo method), 142
- find\_files() (in module datalad.utils), 135
- finish() (datalad.support.annexrepo.ProcessAnnexProgressIndicators method), 151
- FinishPipeline, 162
- format() (datalad.log.ColorFormatter method), 134
- fsck() (datalad.support.annexrepo.AnnexRepo method), 142
- ## G
- generate\_uids() (in module datalad.customremotes.base), 156
- get() (datalad.config.ConfigManager method), 159
- get() (datalad.support.annexrepo.AnnexRepo method), 142
- get() (datalad.support.annexrepo.BatchedAnnexes method), 151
- get() (in module datalad.api), 103
- get\_annexed\_files() (datalad.support.annexrepo.AnnexRepo method), 142
- get\_archive() (datalad.support.archives.ArchivesCache method), 152
- get\_contentlocation() (datalad.customremotes.base.AnnexCustomRemote method), 154
- get\_contentlocation() (datalad.support.annexrepo.AnnexRepo method), 142
- get\_corresponding\_branch() (datalad.support.annexrepo.AnnexRepo method), 143
- get\_dataset\_root() (in module datalad.utils), 135
- get\_description() (datalad.support.annexrepo.AnnexRepo method), 143
- get\_dirhash() (datalad.customremotes.base.AnnexCustomRemote method), 154
- get\_extracted\_file() (datalad.support.archives.ExtractedArchive method), 152
- get\_extracted\_filename() (datalad.support.archives.ExtractedArchive method), 152
- get\_extracted\_files() (datalad.support.archives.ExtractedArchive method), 152
- get\_file\_backend() (datalad.support.annexrepo.AnnexRepo method), 143
- get\_file\_key() (datalad.support.annexrepo.AnnexRepo method), 143
- get\_file\_size() (datalad.support.annexrepo.AnnexRepo method), 143
- get\_file\_url() (datalad.customremotes.archives.ArchiveAnnexCustomRemote method), 157
- get\_func\_kwargs\_doc() (in module datalad.utils), 135
- get\_git\_envirion\_adjusted() (datalad.cmd.GitRunner static method), 132
- get\_git\_version() (in module datalad.config), 161

`get_groupwanted()` (datalad.support.annexrepo.AnnexRepo method), 143  
`get_leading_directory()` (datalad.support.archives.ExtractedArchive method), 152  
`get_logfilename()` (in module datalad.utils), 136  
`get_metadata()` (datalad.support.annexrepo.AnnexRepo method), 144  
`get_most_obscure_supported_name()` (in module datalad.tests.utils), 164  
`get_mtimes_and_digests()` (in module datalad.tests.utils), 165  
`get_path_prefix()` (in module datalad.utils), 136  
`get_preferred_content()` (datalad.support.annexrepo.AnnexRepo method), 144  
`get_remotes()` (datalad.support.annexrepo.AnnexRepo method), 144  
`get_repo_instance()` (in module datalad.cmdline.helpers), 168  
`get_repo_pipeline_config_path()` (in module datalad.crawler.pipeline), 162  
`get_repo_pipeline_script_path()` (in module datalad.crawler.pipeline), 162  
`get_runner()` (in module datalad.cmd), 133  
`get_size_from_key()` (datalad.support.annexrepo.AnnexRepo static method), 144  
`get_status()` (datalad.support.annexrepo.AnnexRepo method), 144  
`get_tempfile_kwargs()` (in module datalad.utils), 136  
`get_timestamp_suffix()` (in module datalad.utils), 136  
`get_toppath()` (datalad.support.annexrepo.AnnexRepo class method), 145  
`get_trace()` (in module datalad.utils), 136  
`get_tracking_branch()` (datalad.support.annexrepo.AnnexRepo method), 145  
`get_URLS()` (datalad.customremotes.base.AnnexCustomRemote method), 154  
`get_urls()` (datalad.support.annexrepo.AnnexRepo method), 145  
`get_value()` (datalad.config.ConfigManager method), 159  
`getbool()` (datalad.config.ConfigManager method), 159  
`getfloat()` (datalad.config.ConfigManager method), 159  
`getIncludes()` (datalad.support.configparserinc.SafeConfigParserWithIncludes static method), 153  
`getint()` (datalad.config.ConfigManager method), 159  
`getpwd()` (in module datalad.utils), 136  
`GIT_ANNEX_MIN_VERSION` (datalad.support.annexrepo.AnnexRepo attribute), 139  
`git_annex_version` (datalad.support.annexrepo.AnnexRepo attribute), 145  
`GitRunner` (class in datalad.cmd), 132

## H

`has_option()` (datalad.config.ConfigManager method), 159  
`has_section()` (datalad.config.ConfigManager method), 159  
`HEADER` (datalad.customremotes.base.AnnexExchangeProtocol attribute), 156  
`heavydebug()` (datalad.customremotes.base.AnnexCustomRemote method), 154  
`HelpAction` (class in datalad.cmdline.helpers), 168

## I

`ignore_nose_capturing_stdout()` (in module datalad.tests.utils), 165  
`info()` (datalad.support.annexrepo.AnnexRepo method), 145  
`init_datalad_remote()` (in module datalad.customremotes.base), 156  
`init_remote()` (datalad.support.annexrepo.AnnexRepo method), 145  
`initiate()` (datalad.customremotes.base.AnnexExchangeProtocol method), 156  
`initiate_pipeline_config()` (in module datalad.crawler.pipeline), 162  
`InnerSubmodule` (class in datalad.tests.utils\_testrepos), 167  
`install()` (in module datalad.api), 105  
`integration()` (in module datalad.tests.utils), 165  
`is_available()` (datalad.support.annexrepo.AnnexRepo method), 145  
`is_crippled_fs()` (datalad.support.annexrepo.AnnexRepo method), 146  
`is_direct_mode()` (datalad.support.annexrepo.AnnexRepo method), 146  
`is_dirty()` (datalad.support.annexrepo.AnnexRepo method), 146  
`is_explicit_path()` (in module datalad.utils), 136  
`is_extracted` (datalad.support.archives.ExtractedArchive attribute), 152  
`is_interactive()` (in module datalad.utils), 136  
`is_managed_branch()` (datalad.support.annexrepo.AnnexRepo method), 146  
`is_under_annex()` (datalad.support.annexrepo.AnnexRepo method), 147  
`is_valid_repo()` (datalad.support.annexrepo.AnnexRepo class method), 147  
`items()` (datalad.config.ConfigManager method), 159

## K

keys() (datalad.config.ConfigManager method), 159  
 knows\_annex() (in module datalad.utils), 136

## L

line\_profile() (in module datalad.utils), 137  
 link\_file\_load() (in module datalad.cmd), 133  
 lmtime() (in module datalad.utils), 137  
 load\_pipeline\_from\_config() (in module datalad.crawler.pipeline), 162  
 load\_pipeline\_from\_module() (in module datalad.crawler.pipeline), 162  
 load\_pipeline\_from\_template() (in module datalad.crawler.pipeline), 163  
 lock() (datalad.support.annexrepo.AnnexRepo method), 147  
 log() (datalad.cmd.Runner method), 132  
 log\_cwd (datalad.cmd.Runner attribute), 133  
 log\_env (datalad.cmd.Runner attribute), 133  
 log\_message() (datalad.tests.utils.SilentHTTPHandler method), 164  
 log\_outputs (datalad.cmd.Runner attribute), 133  
 log\_stdin (datalad.cmd.Runner attribute), 133  
 LogLevelAction (class in datalad.cmdline.helpers), 168  
 ls() (in module datalad.api), 123

## M

main() (datalad.customremotes.base.AnnexCustomRemote method), 154  
 main() (in module datalad.cmdline.main), 168  
 main() (in module datalad.customremotes.archives), 158  
 main() (in module datalad.customremotes.main), 154  
 make\_tempfile() (in module datalad.utils), 137  
 md5sum() (in module datalad.utils), 137  
 merge\_annex() (datalad.support.annexrepo.AnnexRepo method), 147  
 migrate\_backend() (datalad.support.annexrepo.AnnexRepo method), 147

## N

NestedDataset (class in datalad.tests.utils\_testrepos), 167  
 nok\_startswith() (in module datalad.tests.utils), 165  
 not\_supported\_on\_windows() (in module datalad.utils), 137  
 nothing\_cm() (in module datalad.utils), 137

## O

obtain() (datalad.config.ConfigManager method), 159  
 ok\_annex\_get() (in module datalad.tests.utils), 165  
 ok\_archives\_caches() (in module datalad.tests.utils), 165  
 ok\_broken\_symlink() (in module datalad.tests.utils), 165  
 ok\_clean\_git() (in module datalad.tests.utils), 165

ok\_endswith() (in module datalad.tests.utils), 165  
 ok\_exists() (in module datalad.tests.utils), 166  
 ok\_file\_has\_content() (in module datalad.tests.utils), 166  
 ok\_file\_under\_git() (in module datalad.tests.utils), 166  
 ok\_generator() (in module datalad.tests.utils), 166  
 ok\_git\_config\_not\_empty() (in module datalad.tests.utils), 166  
 ok\_good\_symlink() (in module datalad.tests.utils), 166  
 ok\_startswith() (in module datalad.tests.utils), 166  
 ok\_symlink() (in module datalad.tests.utils), 166  
 optional\_args() (in module datalad.utils), 137  
 options() (datalad.config.ConfigManager method), 160

## P

parser\_add\_common\_opt() (in module datalad.cmdline.helpers), 168  
 PATH, 28  
 path (datalad.support.archives.ArchivesCache attribute), 152  
 path (datalad.support.archives.ExtractedArchive attribute), 152  
 path (datalad.tests.utils\_testrepos.InnerSubmodule attribute), 167  
 path (datalad.tests.utils\_testrepos.TestRepo attribute), 167  
 populate() (datalad.tests.utils\_testrepos.BasicAnnexTestRepo method), 167  
 populate() (datalad.tests.utils\_testrepos.BasicGitTestRepo method), 167  
 populate() (datalad.tests.utils\_testrepos.NestedDataset method), 167  
 populate() (datalad.tests.utils\_testrepos.SubmoduleDataset method), 167  
 populate() (datalad.tests.utils\_testrepos.TestRepo method), 167  
 posix\_relpth() (in module datalad.utils), 138  
 precommit() (datalad.support.annexrepo.AnnexRepo method), 147  
 ProcessAnnexProgressIndicators (class in datalad.support.annexrepo), 151  
 progress() (datalad.customremotes.base.AnnexCustomRemote method), 154  
 protocol (datalad.cmd.Runner attribute), 133  
 proxy() (datalad.support.annexrepo.AnnexRepo method), 147  
 publish() (in module datalad.api), 107  
 put\_file\_under\_git() (in module datalad.tests.utils), 166

## R

read() (datalad.customremotes.base.AnnexCustomRemote method), 154  
 read() (datalad.support.configparserinc.SafeConfigParserWithIncludes method), 154

[readline\\_json\(\)](#) (in module `datalad.support.annexrepo`), 151  
[readline\\_rstripped\(\)](#) (in module `datalad.support.annexrepo`), 151  
[readlines\\_until\\_ok\\_or\\_failed\(\)](#) (in module `datalad.support.annexrepo`), 151  
[records\\_callable](#) (`datalad.customremotes.base.AnnexExchangeProtocol` attribute), 156  
[records\\_ext\\_commands](#) (`datalad.customremotes.base.AnnexExchangeProtocol` attribute), 156  
[RegexType](#) (class in `datalad.cmdline.helpers`), 168  
[reload\(\)](#) (`datalad.config.ConfigManager` method), 160  
[remove\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148  
[remove\(\)](#) (in module `datalad.api`), 108  
[remove\\_section\(\)](#) (`datalad.config.ConfigManager` method), 160  
[rename\\_section\(\)](#) (`datalad.config.ConfigManager` method), 160  
[REPO\\_CLASS](#) (`datalad.tests.utils_testrepos.BasicAnnexTestRepo` attribute), 166  
[REPO\\_CLASS](#) (`datalad.tests.utils_testrepos.BasicGitTestRepo` attribute), 167  
[REPO\\_CLASS](#) (`datalad.tests.utils_testrepos.TestRepo` attribute), 167  
[repo\\_info\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148  
[req\\_CHECKPRESENT\(\)](#) (`datalad.customremotes.archives.ArchiveAnnexCustomRemote` method), 157  
[req\\_CHECKPRESENT\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_CHECKURL\(\)](#) (`datalad.customremotes.archives.ArchiveAnnexCustomRemote` method), 157  
[req\\_CHECKURL\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_CLAIMURL\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_GETAVAILABILITY\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_GETCOST\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_INITREMOTE\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_PREPARE\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_REMOVE\(\)](#) (`datalad.customremotes.archives.ArchiveAnnexCustomRemote` method), 157  
[req\\_REMOVE\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_TRANSFER\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[req\\_WHEREIS\(\)](#) (`datalad.customremotes.archives.ArchiveAnnexCustomRemote` method), 158  
[req\\_WHEREIS\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 155  
[reset\\_pipeline\(\)](#) (in module `datalad.crawler.pipeline`), 163  
[RFC](#)  
[RFC 822](#), 26  
[rm\\_url\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148  
[rmtree\(\)](#) (in module `datalad.utils`), 138  
[rmtemp\(\)](#) (in module `datalad.utils`), 138  
[run\(\)](#) (`datalad.cmd.GitRunner` method), 132  
[run\(\)](#) (`datalad.cmd.Runner` method), 133  
[run\\_pipeline\(\)](#) (in module `datalad.crawler.pipeline`), 163  
[run\\_via\\_pbs\(\)](#) (in module `datalad.cmdline.helpers`), 168  
[Runner](#) (class in `datalad.cmd`), 132

## S

[SafeConfigParserWithIncludes](#) (class in `datalad.support.configparserinc`), 153  
[save\(\)](#) (in module `datalad.api`), 109  
[saved\\_generator\(\)](#) (in module `datalad.utils`), 138  
[search\(\)](#) (in module `datalad.api`), 114  
[SECTION\\_NAME](#) (`datalad.support.configparserinc.SafeConfigParserWithIncludes` attribute), 153  
[sections\(\)](#) (`datalad.config.ConfigManager` method), 161  
[send\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 156  
[send\\_unsupported\(\)](#) (`datalad.customremotes.base.AnnexCustomRemote` method), 156  
[set\(\)](#) (`datalad.config.ConfigManager` method), 161  
[set\\_direct\\_mode\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148  
[set\\_groupwanted\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148  
[set\\_metadata\(\)](#) (`datalad.support.annexrepo.AnnexRepo` method), 148

set\_preferred\_content() (datalad.support.annexrepo.AnnexRepo method), 149  
 set\_remote\_url() (datalad.support.annexrepo.AnnexRepo method), 149  
 setup\_exceptionhook() (in module datalad.utils), 138  
 setup\_parser() (in module datalad.cmdline.main), 168  
 setup\_parser() (in module datalad.customremotes.main), 154  
 shortened\_repr() (in module datalad.utils), 138  
 sibling, 28  
 siblings() (in module datalad.api), 124  
 SilentHTTPHandler (class in datalad.tests.utils), 164  
 skip\_httpretty\_on\_problematic\_pythons() (in module datalad.tests.utils), 166  
 skip\_if\_no\_module() (in module datalad.tests.utils), 166  
 skip\_if\_no\_network() (in module datalad.tests.utils), 166  
 skip\_if\_on\_windows() (in module datalad.tests.utils), 166  
 skip\_if\_scrapy\_without\_selector() (in module datalad.tests.utils), 166  
 skip\_if\_url\_is\_not\_available() (in module datalad.tests.utils), 166  
 skip\_ssh() (in module datalad.tests.utils), 166  
 slash\_join() (in module datalad.utils), 138  
 slow() (in module datalad.tests.utils), 166  
 sorted\_files() (in module datalad.utils), 138  
 sshrun() (in module datalad.api), 124  
 stamp\_path (datalad.support.archives.ExtractedArchive attribute), 152  
 STAMP\_SUFFIX (datalad.support.archives.ExtractedArchive attribute), 152  
 start() (datalad.support.annexrepo.ProcessAnnexProgressIndicators method), 151  
 start\_section() (datalad.customremotes.base.AnnexExchangeProtocol method), 156  
 stop() (datalad.customremotes.archives.ArchiveAnnexCustomRemote method), 158  
 stop() (datalad.customremotes.base.AnnexCustomRemote method), 156  
 strip\_arg\_from\_argv() (in module datalad.cmdline.helpers), 168  
 subdataset, 29  
 subdatasets() (in module datalad.api), 127  
 SubmoduleDataset (class in datalad.tests.utils\_testrepos), 167  
 superdataset, 29  
 SUPPORTED\_SCHEMES (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157  
 swallow\_logs() (in module datalad.utils), 138  
 swallow\_outputs() (in module datalad.utils), 139  
 sync() (datalad.support.annexrepo.AnnexRepo method), 149

## T

test() (in module datalad.api), 131  
 TestRepo (class in datalad.tests.utils\_testrepos), 167  
 try\_multiple() (in module datalad.utils), 139

## U

unannex() (datalad.support.annexrepo.AnnexRepo method), 150  
 uninstall() (in module datalad.api), 112  
 unique() (in module datalad.utils), 139  
 unixify\_path() (in module datalad.support.archives), 153  
 unlock() (datalad.support.annexrepo.AnnexRepo method), 150  
 unlock() (in module datalad.api), 113  
 unset() (datalad.config.ConfigManager method), 161  
 untracked\_files (datalad.support.annexrepo.AnnexRepo attribute), 150  
 update() (in module datalad.api), 111  
 updated() (in module datalad.utils), 139  
 url (datalad.tests.utils\_testrepos.InnerSubmodule attribute), 167  
 url (datalad.tests.utils\_testrepos.TestRepo attribute), 167  
 URL\_PREFIX (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157  
 URL\_SCHEME (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 157  
 usecase() (in module datalad.tests.utils), 166  
 uuid (datalad.support.annexrepo.AnnexRepo attribute), 150

## W

WEB\_UUID (datalad.support.annexrepo.AnnexRepo attribute), 140  
 whereis() (datalad.support.annexrepo.AnnexRepo method), 150  
 with\_pathsep() (in module datalad.utils), 139  
 write\_entries() (datalad.customremotes.base.AnnexExchangeProtocol method), 156  
 write\_section() (datalad.customremotes.base.AnnexExchangeProtocol method), 156

## X

xrun\_pipeline() (in module datalad.crawler.pipeline), 163  
 xrun\_pipeline\_steps() (in module datalad.crawler.pipeline), 163