datalad$_{r}evolution Documentation$

DataLad team

Jan 08, 2019

Contents

The extension equips DataLad with some extra commands that enable alternative workflows.

CHAPTER 1

What is in it for users?

If you are looking for a simple solution to data management that can help you track changes in a project, no matter whether it is about data, source code, or both, this package is for you. With just two simple commands, DataLad does all the work for you. Especially people who are not familiar with Git will find this simplicity appealing.

Here is a quick demo. The **first command** creates a dataset. A dataset is essentially a directory that is managed by DataLad and where all content can be tracked. Let's change directories and enter this dataset after it was created.

```
% datalad rev-create myproject
[INFO   ] Creating a new annex repo at /tmp/myproject
create(ok): /tmp/myproject (dataset)
% cd myproject
```

From now on, any change to this directory can be recorded. For this demo, we will copy some very important construction plans for a nice garden bench into this directory. However, we could also use some GUI tool or a script to make a change, it would make no difference to DataLad.

```
% cp /home/me/bench_plan.svg .
```

Whenever one feels like a noteworthy change has been made, or a milestone was reached, the state of the dataset can be recorded with the **second command**.

```
% datalad rev-save
add(ok): bench_plan.svg (file)
save(ok): . (dataset)
action summary:
  add (ok: 1)
  save (ok: 1)
```

The *rev-save* command will discover any change and record it in the dataset. Changes can not only be added files, but any modification, or deletion, even of entire sub-directories – a simple *datalad rev-save* will make a record of it.

These two commands are all that is needed to record changes in a project within DataLad dataset. The resulting dataset is just as capable as any other DataLad dataset. It can be archived, published, used to go back to a particular state of the project and everything else that DataLad supports. Check out the documentation to learn more about its features.

CHAPTER 2

What is in it for developers?

This extension amends the core base classes with functionality that enables command implementation where the state of a dataset is fully inspected first (in a platform-agnostic fashion) and subsequently low-level tools can execute a desired function based on this status report, with no or minimal further queries of the underlying repository. This helps to better isolate developers from the peculiarities of particular platforms and repository modes, and can lead to more compact code and better performance.

Key component is the *rev-status* command (and its repository-level counterparts) that can report the state of a full dataset hierarchy. It works like a simplified *git status*, but is git-annex aware.

```
% datalad rev-status --recursive
    untracked: /tmp/some/directory_untracked (directory)
    untracked: /tmp/some/file_modified (symlink)
    untracked: /tmp/some/link2dir (symlink)
    untracked: /tmp/some/link2subdsroot (symlink)
    untracked: /tmp/some/other (directory)
     modified: /tmp/some/.gitmodules (file)
        added: /tmp/some/link2subdsdir (symlink)
     modified: /tmp/some/subds_modified (dataset)
    untracked: /tmp/some/subds_modified/new_untracked (file)
% datalad -f json_pp rev-status --annex all subdir/annexed_file.txt
{
  "action": "status",
  "backend": "MD5E",
  "bytesize": "5",
  "gitshasum": "a79f797e1ce00f414131f7e84f47049c4c5c5f1a",
  "has_content": true,
  "humansize": "5 B",
  "key": "MD5E-s5--275876e34cf609db118f3d84b799a790.txt",
  "keyname": "275876e34cf609db118f3d84b799a790.txt",
  "mtime": "unknown",
  "objloc": "/tmp/some/.git/annex/objects/7p/gp/MD5E-s5--
↪275876e34cf609db118f3d84b799a790.txt/MD5E-s5--275876e34cf609db118f3d84b799a790.txt",
  "parentds": "/tmp/some",
  "path": "/tmp/some/subdir/annexed_file.txt",
```

(continues on next page)

(continued from previous page)

```
  "refds": "/tmp/some",
  "state": "clean",
  "status": "ok",
  "type": "file"
}
```

CHAPTER 3

API

3.1 High-level API commands

`rev_create`([path, initopts, force, . . .])	Create a new dataset from scratch.
`rev_save`([path, message, dataset, . . .])	Save the current state of a dataset
`rev_status`([path, dataset, annex, . . .])	Report on the state of dataset content.

3.1.1 datalad.api.rev_create

`datalad.api.`**`rev_create`**(*path=None*, *initopts=None*, *force=False*, *description=None*, *dataset=None*, *no_annex=False*, *fake_dates=False*)

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 via *dataset*). 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 existing 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. [Default: None]
- **initopts** – options to pass to **git init**. Options can be given as a list of command line arguments or as a GitPython-style option dictionary. Note that not all options will lead to viable results. For example '–bare' will not yield a repository where DataLad can adjust files in its worktree. [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. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the create operation on. If a dataset is given, a new subdataset will be created in it. [Default: None]
- **no_annex** (*bool, optional*) – if set, a plain Git repository will be created without any annex. [Default: False]
- **fake_dates** (*bool, optional*) – Configure the repository to use fake dates. The date for a new commit will be set to one second later than the latest commit in the repository. This can be used to anonymize dates. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this

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

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

3.1.2 datalad.api.rev_save

datalad.api.**rev_save**(*path=None, message=None, dataset=None, version_tag=None, recursive=False, recursion_limit=None, updated=False, message_file=None, to_git=None*)

Save the current state of a dataset

Saving the state of a dataset records 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.

Examples

Save any content underneath the current directory, without altering any potential subdataset (use –recursive for that):

```
% datalad save .
```

Save any modification of known dataset content, but leave untracked files (e.g. temporary files) untouched:

```
% dataset save -d <path_to_dataset>
```

Tag the most recent saved state of a dataset:

```
% dataset save -d <path_to_dataset> --version-tag bestyet
```

Parameters

- **path** (*sequence of str or None, optional*) – path/name of the dataset component to save. If given, only changes made to those components are recorded in the new state. [Default: None]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to save. [Default: None]
- **version_tag** (*str or None, optional*) – an additional marker for that state. Every dataset that is touched will receive the tag. [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. [Default: None]

- **updated** (*bool, optional*) – if given, only saves previously tracked paths. [Default: False]
- **message_file** (*str or None, optional*) – take the commit message from this file. This flag is mutually exclusive with -m. [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. Use this flag with a simultaneous selection of paths to save. In general, it is better to pre-configure a dataset to track particular paths, file types, or file sizes with either Git or git- annex. See https://git-annex.branchable.com/tips/largefiles/. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top-level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

3.1.3 datalad.api.rev_status

datalad.api.**rev_status**(*path=None, dataset=None, annex=None, untracked='normal', recursive=False, recursion_limit=None*)

Report on the state of dataset content.

This is an analog to *git status* that is simultaneously crippled and more powerful. It is crippled, because it only supports a fraction of the functionality of its counter part and only distinguishes a subset of the states that Git knows about. But it is also more powerful as it can handle status reports for a whole hierarchy of datasets, with the ability to report on a subset of the content (selection of paths) across any number of datasets in the hierarchy.

Path conventions

All reports are guaranteed to use absolute paths that are underneath the given or detected reference dataset, regardless of whether query paths are given as absolute or relative paths (with respect to the working directory, or to the reference dataset, when such a dataset is given explicitly). Moreover, so-called "explicit relative paths" (i.e. paths that start with '.' or '..') are also supported, and are interpreted as relative paths with respect to the current working directory regardless of whether a reference dataset with specified.

When it is necessary to address a subdataset record in a superdataset without causing a status query for the state _within_ the subdataset itself, this can be achieved by explicitly providing a reference dataset and the path to the root of the subdataset like so:

```
datalad rev-status --dataset . subdspath
```

In contrast, when the state of the subdataset within the superdataset is not relevant, a status query for the content of the subdataset can be obtained by adding a trailing path separator to the query path (rsync-like syntax):

```
datalad rev-status --dataset . subdspath/
```

When both aspects are relevant (the state of the subdataset content and the state of the subdataset within the superdataset), both queries can be combined:

```
datalad rev-status --dataset . subdspath subdspath/
```

When performing a recursive status query, both status aspects of subdataset are always included in the report.

Content types

The following content types are distinguished:

- 'dataset' – any top-level dataset, or any subdataset that is properly registered in superdataset
- 'directory' – any directory that does not qualify for type 'dataset'
- 'file' – any file, or any symlink that is placeholder to an annexed file
- 'symlink' – any symlink that is not used as a placeholder for an annexed file

Content states

The following content states are distinguished:

- 'clean'
- 'added'
- 'modified'
- 'deleted'
- 'untracked'

Parameters

- **`path`** (*`sequence of str or None, optional`*) – path to be evaluated. [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 current working directory. [Default: None]
- **annex** (*{None, 'basic', 'availability', 'all'}, optional*) – Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). [Default: None]
- **untracked** (*{'no', 'normal', 'all'}, optional*) – If and how untracked content is reported when comparing a revision to the state of the work tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. [Default: 'normal']
- **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. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top-level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]

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

3.2 Low-level API

`gitrepo`	Amendment of the DataLad *GitRepo* base class
`annexrepo`	Amendment of the DataLad *AnnexRepo* base class
`dataset`	Amendment of the DataLad *Dataset* base class

3.2.1 datalad_revolution.gitrepo

Amendment of the DataLad *GitRepo* base class

`class` `datalad_revolution.gitrepo.`**`RevolutionGitRepo`**(**args*, ***kwargs*)

Bases: `datalad.support.gitrepo.GitRepo`

`diff`(*fr*, *to*, *paths=None*, *untracked='all'*, *ignore_submodules='no'*)

Like status(), but reports changes between to arbitrary revisions

Parameters

- **`fr`** (*`str`*) – Revision specification (anything that Git understands). Passing *None* considers anything in the target state as new.
- **`to`** (*`str or None`*) – Revision specification (anything that Git understands), or None to compare to the state of the work tree.
- **`paths`** (*`list or None`*) – If given, limits the query to the specified paths. To query all paths specify *None*, not an empty list.
- **`untracked`** (*{'no', 'normal', 'all'}*) – If and how untracked content is reported when no *ref* was given: 'no': no untracked files are reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories.
- **`ignore_submodules`** (*{'no', 'other', 'all'}*) –

Returns

Each content item has an entry under its relative path within the repository. Each value is a dictionary with properties:

type Can be 'file', 'symlink', 'dataset', 'directory'

state Can be 'added', 'untracked', 'clean', 'deleted', 'modified'.

Return type dict

`diffstatus`(*fr*, *to*, *paths=None*, *untracked='all'*, *ignore_submodules='no'*, *_cache=None*)

Like diff(), but reports the status of 'clean' content too

`dirty`

`get_content_info`(*paths=None*, *ref=None*, *untracked='all'*)

Get identifier and type information from repository content.

This is simplified front-end for *git ls-files/tree*.

Both commands differ in their behavior when queried about subdataset paths. ls-files will not report anything, ls-tree will report on the subdataset record. This function uniformly follows the behavior of ls-tree (report on the respective subdataset mount).

Parameters

- **`paths`** (`list(patlib.PurePath)`) – Specific paths, relative to the (resolved repository root, to query info for. Paths must be normed to match the reporting done by Git, i.e. no parent dir components (ala "some/../this"). If none are given, info is reported for all content.
- **`ref`** (`gitref or None`) – If given, content information is retrieved for this Git reference (via ls-tree), otherwise content information is produced for the present work tree (via ls-files).
- **`untracked`** (`{'no', 'normal', 'all'}`) – If and how untracked content is reported when no *ref* was given: 'no': no untracked files are reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories.

Returns

Each content item has an entry under its relative path within the repository. Each value is a dictionary with properties:

type Can be 'file', 'symlink', 'dataset', 'directory'

Note that the reported type will not always match the type of content commited to Git, rather it will reflect the nature of the content minus platform/mode-specifics. For example, a symlink to a locked annexed file on Unix will have a type 'file', reported, while a symlink to a file in Git or directory will be of type 'symlink'.

gitshasum SHASUM of the item as tracked by Git, or None, if not tracked. This could be different from the SHASUM of the file in the worktree, if it was modified.

Return type dict

Raises `ValueError` – In case of an invalid Git reference (e.g. 'HEAD' in an empty repository)

`get_staged_paths`()

Returns a list of any stage repository path(s)

This is a rather fast call, as it will not depend on what is going on in the worktree.

`is_dirty`(***kwargs*)

`save`(*message=None*, *paths=None*, *_status=None*, ***kwargs*)

Save dataset content.

Parameters

- **`message`** (`str or None`) – A message to accompany the changeset in the log. If None, a default message is used.
- **`paths`** (`list or None`) – Any content with path matching any of the paths given in this list will be saved. Matching will be performed against the dataset status (GitRepo.status()), or a custom status provided via *_status*. If no paths are provided, ALL non-clean paths present in the repo status or *_status* will be saved.
- **`ignore_submodules`** (`{'no', 'all'}`) – If *_status* is not given, will be passed as an argument to Repo.status(). With 'all' no submodule state will be saved in the dataset. Note that submodule content will never be saved in their respective datasets, as this function's scope is limited to a single dataset.

- **`_status`** (*`dict or None`*) – If None, Repo.status() will be queried for the given *ds*. If a dict is given, its content will be used as a constraint. For example, to save only modified content, but no untracked content, set *paths* to None and provide a *_status* that has no entries for untracked content.
- **`**kwargs`** – Additional arguments that are passed to underlying Repo methods. Supported:
  - git : bool (passed to Repo.add()
  - ignore_submodules : {‘no’, ‘other’, ‘all’} passed to Repo.status()
  - untracked : {‘no’, ‘normal’, ‘all’} - passed to Repo.satus()

`save_`(*message=None*, *paths=None*, *_status=None*, ***kwargs*)

Like *save()* but working as a generator.

`status`(*paths=None*, *untracked='all'*, *ignore_submodules='no'*)

Simplified *git status* equivalent.

Parameters

- **`paths`** (*`list or None`*) – If given, limits the query to the specified paths. To query all paths specify *None*, not an empty list. If a query path points into a subdataset, a report is made on the subdataset record within the queried dataset only (no recursion).
- **`untracked`** (*`{'no', 'normal', 'all'}`*) – If and how untracked content is reported when no *ref* was given: ‘no’: no untracked files are reported; ‘normal’: untracked files and entire untracked directories are reported as such; ‘all’: report individual files even in fully untracked directories.
- **`ignore_submodules`** (*`{'no', 'other', 'all'}`*) –

Returns

Each content item has an entry under its relative path within the repository. Each value is a dictionary with properties:

type Can be ‘file’, ‘symlink’, ‘dataset’, ‘directory’

state Can be ‘added’, ‘untracked’, ‘clean’, ‘deleted’, ‘modified’.

Return type dict

3.2.2 datalad_revolution.annexrepo

Amendment of the DataLad *AnnexRepo* base class

`class` `datalad_revolution.annexrepo.`**`RevolutionAnnexRepo`**(*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*, *fake_dates=False*)

Bases: `datalad.support.annexrepo.AnnexRepo`, *`datalad_revolution.gitrepo.RevolutionGitRepo`*

`annexstatus`(*paths=None*, *untracked='all'*)

`get_content_annexinfo`(*paths=None*, *init='git'*, *ref=None*, *eval_availability=False*, *key_prefix=''*, ***kwargs*)

Parameters

- **`paths`** (*`list`*) – Specific paths to query info for. In none are given, info is reported for all content.
- **`init`** (*`'git' or dict-like or None`*) – If set to 'git' annex content info will ammend the output of GitRepo.get_content_info(), otherwise the dict-like object supplied will receive this information and the present keys will limit the report of annex properties. Alternatively, if *None* is given, no initialization is done, and no limit is in effect.
- **`ref`** (*`gitref or None`*) – If not None, annex content info for this Git reference will be produced, otherwise for the content of the present worktree.
- **`eval_availability`** (*`bool`*) – If this flag is given, evaluate whether the content of any annex'ed file is present in the local annex.
- **`**kwargs`** – Additional arguments for GitRepo.get_content_info(), if *init* is set to 'git'.

Returns

Each content item has an entry under its relative path within the repository. Each value is a dictionary with properties:

type Can be 'file', 'symlink', 'dataset', 'directory'

revision SHASUM is last commit affecting the item, or None, if not tracked.

key Annex key of a file (if an annex'ed file)

bytesize Size of an annexed file in bytes.

has_content Bool whether a content object for this key exists in the local annex (with *eval_availability*)

objloc pathlib.Path of the content object in the local annex, if one is available (with *eval_availability*)

Return type dict

`is_dirty`(***kwargs*)

3.2.3 datalad_revolution.dataset

Amendment of the DataLad *Dataset* base class

class `datalad_revolution.dataset.`**`EnsureDataset`**

Bases: `datalad.distribution.dataset.EnsureDataset`

class `datalad_revolution.dataset.`**`RevolutionDataset`**(*path*)

Bases: `datalad.distribution.dataset.Dataset`

`get_subdatasets`(***kwargs*)

`pathobj`

pathobj for the dataset

`repo`

Get an instance of the version control system/repo for this dataset, or None if there is none yet.

If creating an instance of GitRepo is guaranteed to be really cheap this could also serve as a test whether a repo is present.

Returns

Return type GitRepo

`rev_create`(*initopts=None*, *force=False*, *description=None*, *dataset=None*, *no_annex=False*, *fake_dates=False*)

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 via *dataset*). 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 existing 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. [Default: None]
- **`initopts`** – options to pass to **`git init`**. Options can be given as a list of command line arguments or as a GitPython-style option dictionary. Note that not all options will lead to viable results. For example '–bare' will not yield a repository where DataLad can adjust files in its worktree. [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. [Default: None]
- **`dataset`** (*`Dataset or None, optional`*) – specify the dataset to perform the create operation on. If a dataset is given, a new subdataset will be created in it. [Default: None]
- **`no_annex`** (*`bool, optional`*) – if set, a plain Git repository will be created without any annex. [Default: False]

- **fake_dates** (*bool, optional*) – Configure the repository to use fake dates. The date for a new commit will be set to one second later than the latest commit in the repository. This can be used to anonymize dates. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

rev_diff(*to=None*, *path=None*, *dataset=None*, *annex=None*, *untracked='normal'*, *recursive=False*, *recursion_limit=None*)

Report differences between two states of a dataset (hierarchy)

The two to-be-compared states are given via to –from and –to options. These state identifiers are evaluated in the context of the (specified or detected) dataset. In case of a recursive report on a dataset hierarchy corresponding state pairs for any subdataset are determined from the subdataset record in the respective superdataset. Only changes recorded in a subdataset between these two states are reported, and so on.

Any paths given as additional arguments will be used to constrain the difference report. As with Git's diff, it will not result in an error when a path is specified that does not exist on the filesystem.

Reports are very similar to those of the *rev-status* command, with the distinguished content types and states being identical.

Parameters

- **fr** (*1-item sequence of str, optional*) – original state to compare to, as given by any identifier that Git understands. [Default: 'HEAD']
- **to** (*1-item sequence of str or None, optional*) – state to compare against the original state, as given by any identifier that Git understands. If none is specified, the state of the worktree will be used compared. [Default: None]
- **path** (*sequence of str or None, optional*) – path to contrain the report to. [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 current working directory. [Default: None]
- **annex** (*{None, 'basic', 'availability', 'all'}, optional*) – Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). [Default: None]
- **untracked** (*{'no', 'normal', 'all'}, optional*) – If and how untracked content is reported when comparing a revision to the state of the work tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. [Default: 'normal']
- **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. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]

- **`result_xfm`** (*`{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional`*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **`return_type`** (*`{'generator', 'list', 'item-or-list'}, optional`*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

`rev_run`(*dataset=None*, *inputs=None*, *outputs=None*, *expand=None*, *explicit=False*, *message=None*, *sidecar=None*)

Run an arbitrary shell command and record its impact on a dataset.

It is recommended to craft the command such that it can run in the root directory of the dataset that the command will be recorded in. However, as long as the command is executed somewhere underneath the dataset root, the exact location will be recorded relative to the dataset root.

If the executed command did not alter the dataset in any way, no record of the command execution is made.

If the given command errors, a *CommandError* exception with the same exit code will be raised, and no modifications will be saved.

Command format

A few placeholders are supported in the command via Python format specification. "{pwd}" will be replaced with the full path of the current working directory. "{dspath}" will be replaced with the full path of the dataset that run is invoked on. "{inputs}" and "{outputs}" represent the values specified by *inputs* and *outputs*. If multiple values are specified, the values will be joined by a space. The order of the values will match that order from the command line, with any globs expanded in alphabetical order (like bash). Individual values can be accessed with an integer index (e.g., "{inputs[0]}").

Note that the representation of the inputs or outputs in the formatted command string depends on whether the command is given as a list of arguments or as a string. The concatenated list of inputs or outputs will be surrounded by quotes when the command is given as a list but not when it is given as a string. This means that the string form is required if you need to pass each input as a separate argument to a preceding script (i.e., write the command as "./script {inputs}", quotes included). The string form should also be used if the input or output paths contain spaces or other characters that need to be escaped.

To escape a brace character, double it (i.e., "{{" or "}}").

Custom placeholders can be added as configuration variables under "datalad.run.substitutions". As an example:

Add a placeholder "name" with the value "joe":

```
% git config --file=.datalad/config datalad.run.substitutions.name joe
% datalad add -m "Configure name placeholder" .datalad/config
```

Access the new placeholder in a command:

```
% datalad run "echo my name is {name} >me"
```

Parameters

- **`cmd`** – command for execution. [Default: None]

- **dataset** (*Dataset or None, optional*) – specify the dataset to record the command results in. An attempt is made to identify the dataset based on the current working directory. If a dataset is given, the command will be executed in the root directory of this dataset. [Default: None]
- **inputs** – A dependency for the run. Before running the command, the content of this file will be retrieved. A value of "." means "run **datalad get .**". The value can also be a glob. [Default: None]
- **outputs** – Prepare this file to be an output file of the command. A value of "." means "run **datalad unlock .**" (and will fail if some content isn't present). For any other value, if the content of this file is present, unlock the file. Otherwise, remove it. The value can also be a glob. [Default: None]
- **expand** (*None or {'inputs', 'outputs', 'both'}, optional*) – Expand globs when storing inputs and/or outputs in the commit message. [Default: None]
- **explicit** (*bool, optional*) – Consider the specification of inputs and outputs to be explicit. Don't warn if the repository is dirty, and only save modifications to the listed outputs. [Default: False]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **sidecar** (*None or bool, optional*) – By default, the configuration variable 'datalad.run.record-sidecar' determines whether a record with information on a command's execution is placed into a separate record file instead of the commit message (default: off). This option can be used to override the configured behavior on a case-by-case basis. Sidecar files are placed into the dataset's '.datalad/runinfo' directory (customizable via the 'datalad.run.record-directory' configuration variable). [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this

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

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

rev_save (*message=None*, *dataset=None*, *version_tag=None*, *recursive=False*, *recursion_limit=None*, *updated=False*, *message_file=None*, *to_git=None*)

Save the current state of a dataset

Saving the state of a dataset records 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.

Examples

Save any content underneath the current directory, without altering any potential subdataset (use –recursive for that):

```
% datalad save .
```

Save any modification of known dataset content, but leave untracked files (e.g. temporary files) untouched:

```
% dataset save -d <path_to_dataset>
```

Tag the most recent saved state of a dataset:

```
% dataset save -d <path_to_dataset> --version-tag bestyet
```

Parameters

- **path** (*sequence of str or None, optional*) – path/name of the dataset component to save. If given, only changes made to those components are recorded in the new state. [Default: None]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to save. [Default: None]
- **version_tag** (*str or None, optional*) – an additional marker for that state. Every dataset that is touched will receive the tag. [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. [Default: None]
- **updated** (*bool, optional*) – if given, only saves previously tracked paths. [Default: False]

- **message_file** (*str or None, optional*) – take the commit message from this file. This flag is mutually exclusive with -m. [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. Use this flag with a simultaneous selection of paths to save. In general, it is better to pre-configure a dataset to track particular paths, file types, or file sizes with either Git or git- annex. See https://git-annex.branchable.com/tips/largefiles/. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

rev_status (*dataset=None, annex=None, untracked='normal', recursive=False, recursion_limit=None*)

Report on the state of dataset content.

This is an analog to *git status* that is simultaneously crippled and more powerful. It is crippled, because it only supports a fraction of the functionality of its counter part and only distinguishes a subset of the states that Git knows about. But it is also more powerful as it can handle status reports for a whole hierarchy of datasets, with the ability to report on a subset of the content (selection of paths) across any number of datasets in the hierarchy.

Path conventions

All reports are guaranteed to use absolute paths that are underneath the given or detected reference dataset, regardless of whether query paths are given as absolute or relative paths (with respect to the working directory, or to the reference dataset, when such a dataset is given explicitly). Moreover, so-called "explicit relative paths" (i.e. paths that start with '.' or '..') are also supported, and are interpreted as relative paths with respect to the current working directory regardless of whether a reference dataset with specified.

When it is necessary to address a subdataset record in a superdataset without causing a status query for the state _within_ the subdataset itself, this can be achieved by explicitly providing a reference dataset and the path to the root of the subdataset like so:

```
datalad rev-status --dataset . subdspath
```

In contrast, when the state of the subdataset within the superdataset is not relevant, a status query for the content of the subdataset can be obtained by adding a trailing path separator to the query path (rsync-like syntax):

```
datalad rev-status --dataset . subdspath/
```

When both aspects are relevant (the state of the subdataset content and the state of the subdataset within the superdataset), both queries can be combined:

```
datalad rev-status --dataset . subdspath subdspath/
```

When performing a recursive status query, both status aspects of subdataset are always included in the report.

Content types

The following content types are distinguished:

- 'dataset' – any top-level dataset, or any subdataset that is properly registered in superdataset
- 'directory' – any directory that does not qualify for type 'dataset'
- 'file' – any file, or any symlink that is placeholder to an annexed file
- 'symlink' – any symlink that is not used as a placeholder for an annexed file

Content states

The following content states are distinguished:

- 'clean'
- 'added'
- 'modified'
- 'deleted'
- 'untracked'

Parameters

- **path** (*sequence of str or None, optional*) – path to be evaluated. [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 current working directory. [Default: None]

- **annex** (*{None, 'basic', 'availability', 'all'}, optional*) – Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). [Default: None]
- **untracked** (*{'no', 'normal', 'all'}, optional*) – If and how untracked content is reported when comparing a revision to the state of the work tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. [Default: 'normal']
- **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. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **proc_post** – Like *proc_pre*, but procedures are executed after the main command has finished. [Default: None]
- **proc_pre** – DataLad procedure to run prior to the main command. The argument a list of lists with procedure names and optional arguments. Procedures are called in the order their are given in this list. It is important to provide the respective target dataset to run a procedure on as the *dataset* argument of the main command. [Default: None]
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** (*{'default', 'json', 'json_pp', 'tailored'} or None, optional*) – format of return value rendering on stdout. [Default: None]
- **result_xfm** (*{'relpaths', 'metadata', 'datasets', 'successdatasets-or-none', 'paths'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of

an empty list. [Default: ‘list’]

`datalad_revolution.dataset.`**`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.

If *path* is a symlink or something other than a directory, its the root dataset containing its parent directory will be reported. If none can be found, at a symlink at *path* is pointing to a dataset, *path* itself will be reported as the root.

`datalad_revolution.dataset.`**`path_under_dataset`**`(`*ds*, *path*`)`

`datalad_revolution.dataset.`**`require_dataset`**`(`*dataset*, *check_installed=True*, *purpose=None*`)`

`datalad_revolution.dataset.`**`resolve_path`**`(`*path*, *ds=None*`)`

Resolve a path specification (against a Dataset location)

Any explicit path (absolute or relative) is returned as an absolute path. In case of an explicit relative path (e.g. “./some”, or “.some” on windows), the current working directory is used as reference. Any non-explicit relative path is resolved against as dataset location, i.e. considered relative to the location of the dataset. If no dataset is provided, the current working directory is used.

Note however, that this function is not able to resolve arbitrarily obfuscated path specifications. All operations are purely lexical, and no actual path resolution against the filesystem content is performed. Consequently, common relative path arguments like ‘../something’ (relative to PWD) can be handled properly, but things like ‘down/../under’ cannot, as resolving this path properly depends on the actual target of any (potential) symlink leading up to ‘..’.

Parameters

- **`path`** (*`str or PathLike`*) – Platform-specific path specific path specification.
- **`ds`** (*`Dataset or None`*) – Dataset instance to resolve non-explicit relative paths against.

Returns

Return type *pathlib.Path* object

CHAPTER 4

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). Additional support is provided by the German federal state of Saxony-Anhalt and the European Regional Development Fund (ERDF), Project: Center for Behavioral Brain Sciences, Imaging Platform

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

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index