datalad$_{r}evolutionDocumentation$

DataLad team

Apr 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** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value

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

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

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 rev-save .
```

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

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

Tag the most recent saved state of a dataset:

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

Note: For performance reasons, any Git repository without an initial commit located inside a Dataset is ignored, and content underneath it will be saved to the respective superdataset. DataLad datasets always have an initial commit, hence are not affected by this behavior.

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

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** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary

transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]

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

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

3.2.2 datalad_revolution.annexrepo

Amendment of the DataLad *AnnexRepo* base class

3.2.3 datalad_revolution.dataset

Amendment of the DataLad *Dataset* base class

CHAPTER 4

Acknowledgments

DataLad development is being performed as part of a US-German collaboration in computational neuroscience (CRCNS) 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

d

Index

D

R