Datalad Extension for deprecated functionality

Release 0.3.0+6.g4f29653.dirty

DataLad team

Mar 19, 2024

CONTENTS

CHAPTER

ONE

API

1.1 High-level API commands

ls(loc[, recursive, fast, all_, long_, ...])	List summary information about URLs and dataset(s)
publish([path, dataset, to, since, missing, ...])	Publish a dataset to a known sibling.
metadata([path, dataset, get_aggregates, ...])	Metadata reporting for files and entire datasets
search([query, dataset, force_reindex, ...])	Search dataset metadata
extract_metadata(types[, files, dataset])	Run one or more of DataLad's metadata extractors on a dataset or file.
aggregate_metadata([path, dataset, ...])	Aggregate metadata of one or more datasets for later query.

1.1.1 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://...
- **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. [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]

1.1.2 datalad.api.publish

`datalad.api.`**`publish`**(*path=None*, *dataset=None*, *to=None*, *since=None*, *missing='fail'*, *force=False*, *transfer_data='auto'*, *recursive=False*, *recursion_limit=None*, *git_opts=None*, *annex_opts=None*, *annex_copy_opts=None*, *jobs=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 they support data upload).

Note: This command is deprecated. It will be removed from DataLad eventually, but no earlier than the 0.15 release. The *push* command (new in 0.13.0) provides an alternative interface. Critical differences are that *push* transfers annexed data by default and does not handle sibling creation (i.e. it does not have a *--missing* option).

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. [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. [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). [Default: None]
- **since** (*str or None, optional*) -- specifies commit-ish (tag, shasum, etc.) from which to look for changes to decide whether pushing is necessary. If '^' is given, the last state of

the current branch at the sibling is taken as a starting point. An empty string ('') for the same effect is still supported). [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. [Default: 'fail']
- **force** (*bool, optional*) -- enforce doing publish activities (git push etc) regardless of the analysis if they seemed needed. [Default: False]
- **transfer_data** (*{'auto', 'none', 'all'}, optional*) -- ADDME. [Default: 'auto']
- **recursive** (*bool, optional*) -- if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) -- limit recursion into subdatasets to the given number of levels. [Default: None]
- **git_opts** (*str or None, optional*) -- option string to be passed to git calls. [Default: None]
- **annex_opts** (*str or None, optional*) -- option string to be passed to git annex calls. [Default: None]
- **annex_copy_opts** (*str or None, optional*) -- option string to be passed to git annex copy calls. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) -- how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) -- behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) -- if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** -- select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) -- if given, each to-be-returned result

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

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

1.1.3 datalad.api.metadata

datalad.api.**metadata**(*path=None*, ***, *dataset=None*, *get_aggregates=False*, *reporton='all'*, *recursive=False*)

Metadata reporting for files and entire datasets

Two types of metadata are supported:

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

Both types can be accessed with this command.

Examples

Report the metadata of a single file, as aggregated into the closest locally available dataset, containing the query path:

```
% datalad metadata somedir/subdir/thisfile.dat
```

Sometimes it is helpful to get metadata records formatted in a more accessible form, here as pretty-printed JSON:

```
% datalad -f json_pp metadata somedir/subdir/thisfile.dat
```

Same query as above, but specify which dataset to query (must be containing the query path):

```
% datalad metadata -d . somedir/subdir/thisfile.dat
```

Report any metadata record of any dataset known to the queried dataset:

```
% datalad metadata --recursive --reporton datasets
```

Get a JSON-formatted report of aggregated metadata in a dataset, incl. information on enabled metadata extractors, dataset versions, dataset IDs, and dataset paths:

```
% datalad -f json metadata --get-aggregates
```

Parameters

- **path** (*sequence of str or None, optional*) -- path(s) to query metadata for. [Default: None]
- **dataset** (*Dataset or None, optional*) -- dataset to query. If given, metadata will be reported as stored in this dataset. Otherwise, the closest available dataset containing a query path will be consulted. [Default: None]

- **get_aggregates** (*bool, optional*) -- if set, yields all (sub)datasets for which aggregate metadata are available in the dataset. No other action is performed, even if other arguments are given. The reported results contain a datasets's ID, the commit hash at which metadata aggregation was performed, and the location of the object file(s) containing the aggregated metadata. [Default: False]
- **reporton** (*{'all', 'datasets', 'files', 'none'}, optional*) -- choose on what type result to report on: 'datasets', 'files', 'all' (both datasets and files), or 'none' (no report). [Default: 'all']
- **recursive** (*bool, optional*) -- if set, recurse into potential subdatasets. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) -- behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) -- if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** -- select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) -- if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) -- return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

1.1.4 datalad.api.search

datalad.api.**search**(*query=None*, ***, *dataset=None*, *force_reindex=False*, *max_nresults=None*, *mode=None*, *full_record=False*, *show_keys=None*, *show_query=False*, *metadata_source='legacy'*)

Search dataset metadata

DataLad can search metadata extracted from a dataset and/or aggregated into a superdataset (see the *aggregate-metadata* command). This makes it possible to discover datasets, or individual files in a dataset even when they are not available locally.

Ultimately DataLad metadata are a graph of linked data structures. However, this command does not (yet) support queries that can exploit all information stored in the metadata. At the moment the following search modes are implemented that represent different trade-offs between the expressiveness of a query and the computational and storage resources required to execute a query.

- egrep (default)
- egrepcs [case-sensitive egrep]
- textblob
- autofield

An alternative default mode can be configured by tuning the configuration variable 'datalad.search.default-mode':

```
[datalad "search"]
  default-mode = egrepcs
```

Each search mode has its own default configuration for what kind of documents to query. The respective default can be changed via configuration variables:

```
[datalad "search"]
  index-<mode_name>-documenttype = (all|datasets|files)
```

Mode: egrep/egrepcs

These search modes are largely ignorant of the metadata structure, and simply perform matching of a search pattern against a flat string-representation of metadata. This is advantageous when the query is simple and the metadata structure is irrelevant, or precisely known. Moreover, it does not require a search index, hence results can be reported without an initial latency for building a search index when the underlying metadata has changed (e.g. due to a dataset update). By default, these search modes only consider datasets and do not investigate records for individual files for speed reasons. Search results are reported in the order in which they were discovered.

Queries can make use of Python regular expression syntax (https://docs.python.org/3/library/re.html). In *egrep* mode, matching is case-insensitive when the query does not contain upper case characters, but is case-sensitive when it does. In *egrepcs* mode, matching is always case-sensitive. Expressions will match anywhere in a metadata string, not only at the start.

When multiple queries are given, all queries have to match for a search hit (AND behavior).

It is possible to search individual metadata key/value items by prefixing the query with a metadata key name, separated by a colon (':'). The key name can also be a regular expression to match multiple keys. A query match happens when any value of an item with a matching key name matches the query (OR behavior). See examples for more information.

Examples

Query for (what happens to be) an author:

```
% datalad search haxby
```

Queries are case-INsensitive when the query contains no upper case characters, and can be regular expressions. Use *egrepcs* mode when it is desired to perform a case-sensitive lowercase match:

```
% datalad search --mode egrepcs halchenko.*haxby
```

This search mode performs NO analysis of the metadata content. Therefore queries can easily fail to match. For example, the above query implicitly assumes that authors are listed in alphabetical order. If that is the case (which may or may not be true), the following query would yield NO hits:

```
% datalad search Haxby.*Halchenko
```

The `textblob` search mode represents an alternative that is more robust in such cases.

For more complex queries multiple query expressions can be provided that all have to match to be considered a hit (AND behavior). This query discovers all files (non-default behavior) that match 'bids.type=T1w' AND 'nifti1.qform_code=scanner':

```
% datalad -c datalad.search.index-egrep-documenttype=all search bids.type:T1w␣
↪nifti1.qform_code:scanner
```

Key name selectors can also be expressions, which can be used to select multiple keys or construct "fuzzy" queries. In such cases a query matches when any item with a matching key matches the query (OR behavior). However, multiple queries are always evaluated using an AND conjunction. The following query extends the example above to match any files that have either 'nifti1.qform_code=scanner' or 'nifti1.sform_code=scanner':

```
% datalad -c datalad.search.index-egrep-documenttype=all search bids.type:T1w␣
↪nifti1.(q|s)form_code:scanner
```

Mode: textblob

This search mode is very similar to the `egrep` mode, but with a few key differences. A search index is built from the string-representation of metadata records. By default, only datasets are included in this index, hence the indexing is usually completed within a few seconds, even for hundreds of datasets. This mode uses its own query language (not regular expressions) that is similar to other search engines. It supports logical conjunctions and fuzzy search terms. More information on this is available from the Whoosh project (search engine implementation):

- Description of the Whoosh query language: http://whoosh.readthedocs.io/en/latest/querylang.html)
- Description of a number of query language customizations that are enabled in DataLad, such as, fuzzy term matching: http://whoosh.readthedocs.io/en/latest/parsing.html#common-customizations

Importantly, search hits are scored and reported in order of descending relevance, hence limiting the number of search results is more meaningful than in the 'egrep' mode and can also reduce the query duration.

Examples

Search for (what happens to be) two authors, regardless of the order in which those names appear in the metadata:

```
% datalad search --mode textblob halchenko haxby
```

Fuzzy search when you only have an approximate idea what you are looking for or how it is spelled:

```
% datalad search --mode textblob haxbi~
```

Very fuzzy search, when you are basically only confident about the first two characters and how it sounds approximately (or more precisely: allow for three edits and require matching of the first two characters):

```
% datalad search --mode textblob haksbi~3/2
```

Combine fuzzy search with logical constructs:

```
% datalad search --mode textblob 'haxbi~ AND (hanke OR halchenko)'
```

Mode: autofield

This mode is similar to the 'textblob' mode, but builds a vastly more detailed search index that represents individual metadata variables as individual fields. By default, this search index includes records for datasets and individual fields, hence it can grow very quickly into a huge structure that can easily take an hour or more to build and require more than a GB of storage. However, limiting it to documents on datasets (see above) retains the enhanced expressiveness of queries while dramatically reducing the resource demands.

Examples

List names of search index fields (auto-discovered from the set of indexed datasets) which either have a field starting with "age" or "gender":

```
% datalad search --mode autofield --show-keys name '\.age' '\.gender'
```

Fuzzy search for datasets with an author that is specified in a particular metadata field:

```
% datalad search --mode autofield bids.author:haxbi~ type:dataset
```

Search for individual files that carry a particular description prefix in their 'nifti1' metadata:

```
% datalad search --mode autofield nifti1.description:FSL* type:file
```

Reporting

Search hits are returned as standard DataLad results. On the command line the '--output-format' (or '-f') option can be used to tweak results for further processing.

Examples

Format search hits as a JSON stream (one hit per line):

```
% datalad -f json search haxby
```

Custom formatting: which terms matched the query of particular results. Useful for investigating fuzzy search results:

```
$ datalad -f '{path}: {query_matched}' search --mode autofield bids.author:haxbi~
```

Parameters

- **query** -- query string, supported syntax and features depends on the selected search mode (see documentation). [Default: None]
- **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. [Default: None]
- **force_reindex** (*bool, optional*) -- force rebuilding the search index, even if no change in the dataset's state has been detected, for example, when the index documenttype configuration has changed. [Default: False]
- **max_nresults** (*int or None, optional*) -- maximum number of search results to report. Setting this to 0 will report all search matches. Depending on the mode this can search substantially slower. If not specified, a mode-specific default setting will be used. [Default: None]
- **mode** -- Mode of search index structure and content. See section SEARCH MODES for details. [Default: None]
- **full_record** (*bool, optional*) -- If set, return the full metadata record for each search hit. Depending on the search mode this might require additional queries. By default, only data that is available to the respective search modes is returned. This always includes essential information, such as the path and the type. [Default: False]
- **show_keys** -- if given, a list of known search keys is shown. If 'name' - only the name is printed one per line. If 'short' or 'full', statistics (in how many datasets, and how many unique values) are printed. 'short' truncates the listing of unique values. QUERY, if provided, is regular expressions any of which keys should contain. No other action is performed (except for reindexing), even if other arguments are given. Each key is accompanied by a term definition in parenthesis (TODO). In most cases a definition is given in the form of a URL. If an ontology definition for a term is known, this URL can resolve to a webpage that provides a comprehensive definition of the term. However, for speed reasons term resolution is solely done on information contained in a local dataset's metadata, and definition URLs might be outdated or point to no longer existing resources. [Default: None]
- **show_query** (*bool, optional*) -- if given, the formal query that was generated from the given query string is shown, but not actually executed. This is mostly useful for debugging purposes. [Default: False]
- **metadata_source** -- if given, defines which metadata source will be used to search. 'legacy' will limit search to metadata in the old format, i.e. stored in '$DATASET/.datalad/metadata'. 'gen4' will limit search to metadata stored by the git-backend of 'datalad-metadata-model'. If 'all' is given, metadata from all supported sources will be included in the search. The default is 'legacy'. [Default: 'legacy']

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

1.1.5 datalad.api.extract_metadata

datalad.api.**extract_metadata**(*types*, *files=None*, *dataset=None*)

Run one or more of DataLad's metadata extractors on a dataset or file.

The result(s) are structured like the metadata DataLad would extract during metadata aggregation. There is one result per dataset/file.

Examples

Extract metadata with two extractors from a dataset in the current directory and also from all its files:

```
$ datalad extract-metadata -d . --type frictionless_datapackage --type datalad_core
```

Extract XMP metadata from a single PDF that is not part of any dataset:

```
$ datalad extract-metadata --type xmp Downloads/freshfromtheweb.pdf
```

Parameters

- **`types`** -- Name of a metadata extractor to be executed.
- **`files`** (*`sequence of str or None, optional`*) -- Path of a file to extract metadata from. [Default: None]
- **`dataset`** (*`Dataset or None, optional`*) -- "Dataset to extract metadata from. If no *file* is given, metadata is extracted from all files of the dataset. [Default: None]
- **`on_failure`** (*`{'ignore', 'continue', 'stop'}, optional`*) -- behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **`result_filter`** (*`callable or None, optional`*) -- if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **`result_renderer`** -- select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **`result_xfm`** (*`{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional`*) -- if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **`return_type`** (*`{'generator', 'list', 'item-or-list'}, optional`*) -- return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

1.1.6 datalad.api.aggregate_metadata

datalad.api.**aggregate_metadata**(*path=None*, ***, *dataset=None*, *recursive=False*, *recursion_limit=None*, *update_mode='target'*, *incremental=False*, *force_extraction=False*, *save=True*)

Aggregate metadata of one or more datasets for later query.

Metadata aggregation refers to a procedure that extracts metadata present in a dataset into a portable representation that is stored a single standardized format. Moreover, metadata aggregation can also extract metadata in this format from one dataset and store it in another (super)dataset. Based on such collections of aggregated metadata it is possible to discover particular datasets and specific parts of their content, without having to obtain the target datasets first (see the DataLad 'search' command).

To enable aggregation of metadata that are contained in files of a dataset, one has to enable one or more metadata extractor for a dataset. DataLad supports a number of common metadata standards, such as the Exchangeable Image File Format (EXIF), Adobe's Extensible Metadata Platform (XMP), and various audio file metadata systems like ID3. DataLad extension packages can provide metadata data extractors for additional metadata sources. For example, the neuroimaging extension provides extractors for scientific (meta)data standards like BIDS, DICOM, and NIfTI1. Some metadata extractors depend on particular 3rd-party software. The list of metadata extractors available to a particular DataLad installation is reported by the 'wtf' command ('datalad wtf').

Enabling a metadata extractor for a dataset is done by adding its name to the 'datalad.metadata.nativetype' configuration variable -- typically in the dataset's configuration file (.datalad/config), e.g.:

```
[datalad "metadata"]
  nativetype = exif
  nativetype = xmp
```

If an enabled metadata extractor is not available in a particular DataLad installation, metadata extraction will not succeed in order to avoid inconsistent aggregation results.

Enabling multiple extractors is supported. In this case, metadata are extracted by each extractor individually, and stored alongside each other. Metadata aggregation will also extract DataLad's own metadata (extractors 'datalad_core', and 'annex').

Metadata aggregation can be performed recursively, in order to aggregate all metadata across all subdatasets, for example, to be able to search across any content in any dataset of a collection. Aggregation can also be performed for subdatasets that are not available locally. In this case, pre-aggregated metadata from the closest available superdataset will be considered instead.

Depending on the versatility of the present metadata and the number of dataset or files, aggregated metadata can grow prohibitively large. A number of configuration switches are provided to mitigate such issues.

datalad.metadata.aggregate-content-<extractor-name>
If set to false, content metadata aggregation will not be performed for the named metadata extractor (a potential underscore '_' in the extractor name must be replaced by a dash '-'). This can substantially reduce the runtime for metadata extraction, and also reduce the size of the generated metadata aggregate. Note, however, that some extractors may not produce any metadata when this is disabled, because their metadata might come from individual file headers only. 'datalad.metadata.store-aggregate-content' might be a more appropriate setting in such cases.

datalad.metadata.aggregate-ignore-fields
Any metadata key matching any regular expression in this configuration setting is removed prior to generating the dataset-level metadata summary (keys and their unique values across all dataset content), and from the dataset metadata itself. This switch can also be used to filter out sensitive information prior aggregation.

datalad.metadata.generate-unique-<extractor-name>
If set to false, DataLad will not auto-generate a summary of unique content metadata values for a partic-

ular extractor as part of the dataset-global metadata (a potential underscore '_' in the extractor name must be replaced by a dash '-'). This can be useful if such a summary is bloated due to minor uninformative (e.g. numerical) differences, or when a particular extractor already provides a carefully designed content metadata summary.

datalad.metadata.maxfieldsize
Any metadata value that exceeds the size threshold given by this configuration setting (in bytes/characters) is removed.

datalad.metadata.store-aggregate-content
If set, extracted content metadata are still used to generate a dataset-level summary of present metadata (all keys and their unique values across all files in a dataset are determined and stored as part of the dataset-level metadata aggregate, see datalad.metadata.generate-unique-<extractor-name>), but metadata on individual files are not stored. This switch can be used to avoid prohibitively large metadata files. Discovery of datasets containing content matching particular metadata properties will still be possible, but such datasets would have to be obtained first in order to discover which particular files in them match these properties.

Parameters

- **path** (*sequence of str or None, optional*) -- path to datasets that shall be aggregated. When a given path is pointing into a dataset, the metadata of the containing dataset will be aggregated. If no paths given, current dataset metadata is aggregated. [Default: None]
- **dataset** (*Dataset or None, optional*) -- topmost dataset metadata will be aggregated into. All dataset between this dataset and any given path will receive updated aggregated metadata from all given paths. [Default: None]
- **recursive** (*bool, optional*) -- if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) -- limit recursion into subdatasets to the given number of levels. [Default: None]
- **update_mode** (*{'all', 'target'}, optional*) -- which datasets to update with newly aggregated metadata: all datasets from any leaf dataset to the top-level target dataset including all intermediate datasets (all), or just the top-level target dataset (target). [Default: 'target']
- **incremental** (*bool, optional*) -- If set, all information on metadata records of subdatasets that have not been (re-)aggregated in this run will be kept unchanged. This is useful when (re-)aggregation only a subset of a dataset hierarchy, for example, because not all subdatasets are locally available. [Default: False]
- **force_extraction** (*bool, optional*) -- If set, all enabled extractors will be engaged regardless of whether change detection indicates that metadata has already been extracted for a given dataset state. [Default: False]
- **save** (*bool, optional*) -- by default all modifications to a dataset are immediately saved. Giving this option will disable this behavior. [Default: True]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) -- behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) -- if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not

evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]

- **result_renderer** -- select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) -- if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) -- return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

1.2 Command line reference

1.2.1 datalad ls

Synopsis

```
datalad ls [-h] [-r] [-F] [-a] [-L] [--config-file CONFIG_FILE] [--list-content {None,
↪first10,md5,full}] [--json {file,display,delete}] [--version] [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 or value must be NONE

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

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

-r, --recursive

recurse into subdirectories.

-F, --fast

only perform fast operations. Would be overridden by --all.

-a, --all

list all (versions of) entries, not e.g. only latest entries in case of S3.

-L, --long

list more information on entries (e.g. acl, urls in s3, annex sizes etc).

--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 or value must be 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.

--version

show the module and its version which provides the command

Authors

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

1.2.2 datalad publish

Synopsis

```
datalad publish [-h] [-d DATASET] [--to LABEL] [--since SINCE] [--missing MODE] [-f] [--
↪transfer-data {auto|none|all}] [-r] [-R LEVELS] [--git-opts STRING] [--annex-opts␣
↪STRING] [--annex-copy-opts STRING] [-J NJOBS] [--version] [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 they support data upload).

NOTE
: This command is deprecated. It will be removed from DataLad eventually, but no earlier than the 0.15 release. The PUSH command (new in 0.13.0) provides an alternative interface. Critical differences are that *push* transfers annexed data by default and does not handle sibling creation (i.e. it does not have a *--missing* option).

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 or value must be NONE

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

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

-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) or value must be 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 or value must be NONE

--since *SINCE*

specifies commit-ish (tag, shasum, etc.) from which to look for changes to decide whether pushing is necessary. If '^' is given, the last state of the current branch at the sibling is taken as a starting point. An empty string ('') for the same effect is still supported). Constraints: value must be a string or value must be 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.

--transfer-data {auto|none|all}

ADDME. Constraints: value must be one of ('auto', 'none', 'all') [Default: 'auto']

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

limit recursion into subdatasets to the given number of levels. Constraints: value must be convertible to type 'int' or value must be NONE

--git-opts STRING

option string to be passed to git calls. Constraints: value must be a string or value must be NONE

--annex-opts STRING

option string to be passed to git annex calls. Constraints: value must be a string or value must be NONE

--annex-copy-opts STRING

option string to be passed to git annex copy calls. Constraints: value must be a string or value must be NONE

-J NJOBS, --jobs NJOBS

how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. Constraints: value must be convertible to type 'int' or value must be NONE or value must be one of ('auto',)

--version

show the module and its version which provides the command

Authors

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

1.2.3 datalad metadata

Synopsis

```
datalad metadata [-h] [-d DATASET] [--get-aggregates] [--reporton TYPE] [-r] [--version]␣
↪[PATH ...]
```

Description

Metadata reporting for files and entire datasets

Two types of metadata are supported:

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

Both types can be accessed with this command.

Examples:

Report the metadata of a single file, as aggregated into the closest locally available dataset, containing the query path:

```
% datalad metadata somedir/subdir/thisfile.dat
```

Sometimes it is helpful to get metadata records formatted in a more accessible form, here as pretty-printed JSON:

```
% datalad -f json_pp metadata somedir/subdir/thisfile.dat
```

Same query as above, but specify which dataset to query (must be containing the query path):

```
% datalad metadata -d . somedir/subdir/thisfile.dat
```

Report any metadata record of any dataset known to the queried dataset:

```
% datalad metadata --recursive --reporton datasets
```

Get a JSON-formatted report of aggregated metadata in a dataset, incl. information on enabled metadata extractors, dataset versions, dataset IDs, and dataset paths:

```
% datalad -f json metadata --get-aggregates
```

Options

PATH

path(s) to query metadata for. Constraints: value must be a string or value must be NONE

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

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

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

dataset to query. If given, metadata will be reported as stored in this dataset. Otherwise, the closest available dataset containing a query path will be consulted. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--get-aggregates

if set, yields all (sub)datasets for which aggregate metadata are available in the dataset. No other action is performed, even if other arguments are given. The reported results contain a datasets's ID, the commit hash at which metadata aggregation was performed, and the location of the object file(s) containing the aggregated metadata.

--reporton TYPE

choose on what type result to report on: 'datasets', 'files', 'all' (both datasets and files), or 'none' (no report). Constraints: value must be one of ('all', 'datasets', 'files', 'none') [Default: 'all']

-r, --recursive

if set, recurse into potential subdatasets.

--version

show the module and its version which provides the command

Authors

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

1.2.4 datalad extract-metadata

Synopsis

```
datalad extract-metadata [-h] --type NAME [-d DATASET] [--version] [FILE ...]
```

Description

Run one or more of DataLad's metadata extractors on a dataset or file.

The result(s) are structured like the metadata DataLad would extract during metadata aggregation. There is one result per dataset/file.

Examples:

Extract metadata with two extractors from a dataset in the current directory and also from all its files:

```
$ datalad extract-metadata -d . --type frictionless_datapackage --type datalad_
↪core
```

Extract XMP metadata from a single PDF that is not part of any dataset:

```
$ datalad extract-metadata --type xmp Downloads/freshfromtheweb.pdf
```

Options

FILE

Path of a file to extract metadata from. Constraints: value must be a string or value must be NONE

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

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

--type NAME

Name of a metadata extractor to be executed. This option can be given more than once.

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

"Dataset to extract metadata from. If no FILE is given, metadata is extracted from all files of the dataset. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--version

show the module and its version which provides the command

Authors

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

1.2.5 datalad aggregate-metadata

Synopsis

```
datalad aggregate-metadata [-h] [-d DATASET] [-r] [-R LEVELS] [--update-mode {all|target}
↪] [--incremental] [--force-extraction] [--nosave] [--version] [PATH ...]
```

Description

Aggregate metadata of one or more datasets for later query.

Metadata aggregation refers to a procedure that extracts metadata present in a dataset into a portable representation that is stored a single standardized format. Moreover, metadata aggregation can also extract metadata in this format from one dataset and store it in another (super)dataset. Based on such collections of aggregated metadata it is possible to discover particular datasets and specific parts of their content, without having to obtain the target datasets first (see the DataLad 'search' command).

To enable aggregation of metadata that are contained in files of a dataset, one has to enable one or more metadata extractor for a dataset. DataLad supports a number of common metadata standards, such as the Exchangeable Image File Format (EXIF), Adobe's Extensible Metadata Platform (XMP), and various audio file metadata systems like ID3. DataLad extension packages can provide metadata data extractors for additional metadata sources. For example, the neuroimaging extension provides extractors for scientific (meta)data standards like BIDS, DICOM, and NIfTI1. Some metadata extractors depend on particular 3rd-party software. The list of metadata extractors available to a particular DataLad installation is reported by the 'wtf' command ('datalad wtf').

Enabling a metadata extractor for a dataset is done by adding its name to the 'datalad.metadata.nativetype' configuration variable -- typically in the dataset's configuration file (.datalad/config), e.g.:

```
[datalad "metadata"]
  nativetype = exif
  nativetype = xmp
```

If an enabled metadata extractor is not available in a particular DataLad installation, metadata extraction will not succeed in order to avoid inconsistent aggregation results.

Enabling multiple extractors is supported. In this case, metadata are extracted by each extractor individually, and stored alongside each other. Metadata aggregation will also extract DataLad's own metadata (extractors 'datalad_core', and 'annex').

Metadata aggregation can be performed recursively, in order to aggregate all metadata across all subdatasets, for example, to be able to search across any content in any dataset of a collection. Aggregation can also be performed for subdatasets that are not available locally. In this case, pre-aggregated metadata from the closest available superdataset will be considered instead.

Depending on the versatility of the present metadata and the number of dataset or files, aggregated metadata can grow prohibitively large. A number of configuration switches are provided to mitigate such issues.

datalad.metadata.aggregate-content-<extractor-name>
If set to false, content metadata aggregation will not be performed for the named metadata extractor (a potential underscore '_' in the extractor name must be replaced by a dash '-'). This can substantially reduce the runtime for metadata extraction, and also reduce the size of the generated metadata aggregate. Note, however, that some

extractors may not produce any metadata when this is disabled, because their metadata might come from individual file headers only. 'datalad.metadata.store-aggregate-content' might be a more appropriate setting in such cases.

datalad.metadata.aggregate-ignore-fields
Any metadata key matching any regular expression in this configuration setting is removed prior to generating the dataset-level metadata summary (keys and their unique values across all dataset content), and from the dataset metadata itself. This switch can also be used to filter out sensitive information prior aggregation.

datalad.metadata.generate-unique-<extractor-name>
If set to false, DataLad will not auto-generate a summary of unique content metadata values for a particular extractor as part of the dataset-global metadata (a potential underscore '_' in the extractor name must be replaced by a dash '-'). This can be useful if such a summary is bloated due to minor uninformative (e.g. numerical) differences, or when a particular extractor already provides a carefully designed content metadata summary.

datalad.metadata.maxfieldsize
Any metadata value that exceeds the size threshold given by this configuration setting (in bytes/characters) is removed.

datalad.metadata.store-aggregate-content
If set, extracted content metadata are still used to generate a dataset-level summary of present metadata (all keys and their unique values across all files in a dataset are determined and stored as part of the dataset-level metadata aggregate, see datalad.metadata.generate-unique-<extractor-name>), but metadata on individual files are not stored. This switch can be used to avoid prohibitively large metadata files. Discovery of datasets containing content matching particular metadata properties will still be possible, but such datasets would have to be obtained first in order to discover which particular files in them match these properties.

Options

PATH

path to datasets that shall be aggregated. When a given path is pointing into a dataset, the metadata of the containing dataset will be aggregated. If no paths given, current dataset metadata is aggregated. Constraints: value must be a string or value must be NONE

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

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

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

topmost dataset metadata will be aggregated into. All dataset between this dataset and any given path will receive updated aggregated metadata from all given paths. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

limit recursion into subdatasets to the given number of levels. Constraints: value must be convertible to type 'int' or value must be NONE

--update-mode {all|target}

which datasets to update with newly aggregated metadata: all datasets from any leaf dataset to the top-level target dataset including all intermediate datasets (all), or just the top-level target dataset (target). Constraints: value must be one of ('all', 'target') [Default: 'target']

--incremental

If set, all information on metadata records of subdatasets that have not been (re-)aggregated in this run will be kept unchanged. This is useful when (re-)aggregation only a subset of a dataset hierarchy, for example, because not all subdatasets are locally available.

--force-extraction

If set, all enabled extractors will be engaged regardless of whether change detection indicates that metadata has already been extracted for a given dataset state.

--nosave

by default all modifications to a dataset are immediately saved. Giving this option will disable this behavior.

--version

show the module and its version which provides the command

Authors

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

1.2.6 datalad search

Synopsis

```
datalad search [-h] [-d DATASET] [--reindex] [--max-nresults MAX_NRESULTS] [--mode
↪{egrep,textblob,autofield}] [--full-record] [--show-keys {name,short,full}] [--show-
↪query] [--metadata-source {legacy,gen4,all}] [--version] [QUERY ...]
```

Description

Search dataset metadata

DataLad can search metadata extracted from a dataset and/or aggregated into a superdataset (see the *aggregate-metadata* command). This makes it possible to discover datasets, or individual files in a dataset even when they are not available locally.

Ultimately DataLad metadata are a graph of linked data structures. However, this command does not (yet) support queries that can exploit all information stored in the metadata. At the moment the following search modes are implemented that represent different trade-offs between the expressiveness of a query and the computational and storage resources required to execute a query.

- egrep (default)
- egrepcs [case-sensitive egrep]
- textblob
- autofield

An alternative default mode can be configured by tuning the configuration variable 'datalad.search.default-mode':

```
[datalad "search"]
  default-mode = egrepcs
```

Each search mode has its own default configuration for what kind of documents to query. The respective default can be changed via configuration variables:

```
[datalad "search"]
  index-<mode_name>-documenttype = (all|datasets|files)
```

Mode: egrep/egrepcs

These search modes are largely ignorant of the metadata structure, and simply perform matching of a search pattern against a flat string-representation of metadata. This is advantageous when the query is simple and the metadata structure is irrelevant, or precisely known. Moreover, it does not require a search index, hence results can be reported without an initial latency for building a search index when the underlying metadata has changed (e.g. due to a dataset update). By default, these search modes only consider datasets and do not investigate records for individual files for speed reasons. Search results are reported in the order in which they were discovered.

Queries can make use of Python regular expression syntax (https://docs.python.org/3/library/re.html). In EGREP mode, matching is case-insensitive when the query does not contain upper case characters, but is case-sensitive when it does. In *egrepcs* mode, matching is always case-sensitive. Expressions will match anywhere in a metadata string, not only at the start.

When multiple queries are given, all queries have to match for a search hit (AND behavior).

It is possible to search individual metadata key/value items by prefixing the query with a metadata key name, separated by a colon (':'). The key name can also be a regular expression to match multiple keys. A query match happens when any value of an item with a matching key name matches the query (OR behavior). See examples for more information.

Examples:

Query for (what happens to be) an author:

```
% datalad search haxby
```

Queries are case-INsensitive when the query contains no upper case characters, and can be regular expressions. Use EGREPCS mode when it is desired to perform a case-sensitive lowercase match:

```
% datalad search --mode egrepcs halchenko.*haxby
```

This search mode performs NO analysis of the metadata content. Therefore queries can easily fail to match. For example, the above query implicitly assumes that authors are listed in alphabetical order. If that is the case (which may or may not be true), the following query would yield NO hits:

```
% datalad search Haxby.*Halchenko
```

The `textblob` search mode represents an alternative that is more robust in such cases.

For more complex queries multiple query expressions can be provided that all have to match to be considered a hit (AND behavior). This query discovers all files (non-default behavior) that match 'bids.type=T1w' AND 'nifti1.qform_code=scanner':

```
% datalad -c datalad.search.index-egrep-documenttype=all search bids.type:T1w␣
↪nifti1.qform_code:scanner
```

Key name selectors can also be expressions, which can be used to select multiple keys or construct "fuzzy" queries. In such cases a query matches when any item with a matching key matches the query (OR behavior). However, multiple queries are always evaluated using an AND conjunction. The following query extends the example above to match any files that have either 'nifti1.qform_code=scanner' or 'nifti1.sform_code=scanner':

```
% datalad -c datalad.search.index-egrep-documenttype=all search bids.type:T1w␣
↪nifti1.(q|s)form_code:scanner
```

Mode: textblob

This search mode is very similar to the `egrep` mode, but with a few key differences. A search index is built from the string-representation of metadata records. By default, only datasets are included in this index, hence the indexing is usually completed within a few seconds, even for hundreds of datasets. This mode uses its own query language (not regular expressions) that is similar to other search engines. It supports logical conjunctions and fuzzy search terms. More information on this is available from the Whoosh project (search engine implementation):

- Description of the Whoosh query language: http://whoosh.readthedocs.io/en/latest/querylang.html)
- Description of a number of query language customizations that are enabled in DataLad, such as, fuzzy term matching: http://whoosh.readthedocs.io/en/latest/parsing.html#common-customizations

Importantly, search hits are scored and reported in order of descending relevance, hence limiting the number of search results is more meaningful than in the 'egrep' mode and can also reduce the query duration.

Examples:

Search for (what happens to be) two authors, regardless of the order in which those names appear in the metadata:

```
% datalad search --mode textblob halchenko haxby
```

Fuzzy search when you only have an approximate idea what you are looking for or how it is spelled:

```
% datalad search --mode textblob haxbi~
```

Very fuzzy search, when you are basically only confident about the first two characters and how it sounds approximately (or more precisely: allow for three edits and require matching of the first two characters):

```
% datalad search --mode textblob haksbi~3/2
```

Combine fuzzy search with logical constructs:

```
% datalad search --mode textblob 'haxbi~ AND (hanke OR halchenko)'
```

Mode: autofield

This mode is similar to the 'textblob' mode, but builds a vastly more detailed search index that represents individual metadata variables as individual fields. By default, this search index includes records for datasets and individual fields, hence it can grow very quickly into a huge structure that can easily take an hour or more to build and require more than a GB of storage. However, limiting it to documents on datasets (see above) retains the enhanced expressiveness of queries while dramatically reducing the resource demands.

Examples:

List names of search index fields (auto-discovered from the set of indexed datasets) which either have a field starting with "age" or "gender":

```
% datalad search --mode autofield --show-keys name '\.age' '\.gender'
```

Fuzzy search for datasets with an author that is specified in a particular metadata field:

```
% datalad search --mode autofield bids.author:haxbi~ type:dataset
```

Search for individual files that carry a particular description prefix in their 'nifti1' metadata:

```
% datalad search --mode autofield nifti1.description:FSL* type:file
```

Reporting

Search hits are returned as standard DataLad results. On the command line the '--output-format' (or '-f') option can be used to tweak results for further processing.

Examples:

Format search hits as a JSON stream (one hit per line):

```
% datalad -f json search haxby
```

Custom formatting: which terms matched the query of particular results. Useful for investigating fuzzy search results:

```
$ datalad -f '{path}: {query_matched}' search --mode autofield bids.
↪author:haxbi~
```

Options

QUERY

query string, supported syntax and features depends on the selected search mode (see documentation).

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

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

-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) or value must be NONE

--reindex

force rebuilding the search index, even if no change in the dataset's state has been detected, for example, when the index documenttype configuration has changed.

--max-nresults *MAX_NRESULTS*

maximum number of search results to report. Setting this to 0 will report all search matches. Depending on the mode this can search substantially slower. If not specified, a mode-specific default setting will be used. Constraints: value must be convertible to type 'int' or value must be NONE

--mode {egrep,textblob,autofield}

Mode of search index structure and content. See section SEARCH MODES for details.

--full-record, -f

If set, return the full metadata record for each search hit. Depending on the search mode this might require additional queries. By default, only data that is available to the respective search modes is returned. This always includes essential information, such as the path and the type.

--show-keys {name,short,full}

if given, a list of known search keys is shown. If 'name' - only the name is printed one per line. If 'short' or 'full', statistics (in how many datasets, and how many unique values) are printed. 'short' truncates the listing of unique values. QUERY, if provided, is regular expressions any of which keys should contain. No other action is performed (except for reindexing), even if other arguments are given. Each key is accompanied by a term definition in parenthesis (TODO). In most cases a definition is given in the form of a URL. If an ontology definition for a term is known, this URL can resolve to a webpage that provides a comprehensive definition of the term. However, for speed reasons term resolution is solely done on information contained in a local dataset's metadata, and definition URLs might be outdated or point to no longer existing resources.

--show-query

if given, the formal query that was generated from the given query string is shown, but not actually executed. This is mostly useful for debugging purposes.

--metadata-source {legacy,gen4,all}

if given, defines which metadata source will be used to search. 'legacy' will limit search to metadata in the old format, i.e. stored in '$DATASET/.datalad/metadata'. 'gen4' will limit search to metadata stored by the git-backend of 'datalad-metadata-model'. If 'all' is given, metadata from all supported sources will be included in the search. The default is 'legacy'. [Default: 'legacy']

--version

show the module and its version which provides the command

Authors

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

1.3 Miscellaneous functionality

auto.AutomagicIO([autoget, activate, check_once])	Class to proxy commonly used API for accessing files so they get automatically fetched

1.3.1 datalad_deprecated.auto.AutomagicIO

class `datalad_deprecated.auto.`**`AutomagicIO`**(*autoget=True*, *activate=False*, *check_once=False*)

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

`__init__`(*autoget=True*, *activate=False*, *check_once=False*)

Parameters

- **autoget** --
- **activate** --
- **check_once** (*bool, optional*) -- To speed things up and avoid unnecessary repeated checks, if True, paths considered for proxying and corresponding repositories are remembered, and are not subject to datalad checks on subsequent calls. This option is to be used if you do not expect new git repositories to not be created and files not to get dropped while operating under AutomagicIO supervision.

Methods

__init__([autoget, activate, check_once])	**param autoget**
activate()	
deactivate()	

Attributes

active
autoget

CHAPTER TWO

INDICES AND TABLES

- genindex
- search

INDEX