Datalad Next

Release 1.0.0b3+25.g0cb44b0.dirty

DataLad team

Jul 03, 2023
## CONTENTS

1 Installation and usage

2 API
   2.1 High-level API commands ........................................ 5
   2.2 Command line reference ........................................... 18
   2.3 Python tooling ..................................................... 31
   2.4 Git remote helpers ............................................... 71
   2.5 Git-annex backends ............................................... 76
   2.6 Git-annex special remotes ....................................... 81
   2.7 DataLad patches .................................................. 89
   2.8 Developer Guide .................................................. 94

3 Indices and tables

   Python Module Index .................................................. 99

   Index ........................................................................ 101
This DataLad extension can be thought of as a staging area for additional functionality, or for improved performance and user experience. Unlike other topical or more experimental extensions, the focus here is on functionality with broad applicability. This extension is a suitable dependency for other software packages that intend to build on this improved set of functionality.
INSTALLATION AND USAGE

Install from PyPi or Github like any other Python package:

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

Once installed, additional commands provided by this extension are immediately available. However, in order to fully benefit from all improvements, the extension has to be enabled for auto-loading by executing:

```
git config --global --add datalad.extensions.load next
```

Doing so will enable the extension to also alter the behavior the core DataLad package and its commands.
CHAPTER TWO

API

2.1 High-level API commands

2.1.1 High-level API commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_sibling_webdav(url, *[dataset, ...])</td>
<td>Create a sibling(-tandem) on a WebDAV server</td>
</tr>
<tr>
<td>credentials([action, spec, name, prompt, ...])</td>
<td>Credential management and query</td>
</tr>
<tr>
<td>download(spec, *[dataset, force, ...])</td>
<td>Download from URLs</td>
</tr>
<tr>
<td>ls_file_collection(type, collection, *[hash])</td>
<td>Report information on files in a collection</td>
</tr>
<tr>
<td>tree([path, depth, recursive, ...])</td>
<td>Visualize directory and dataset hierarchies</td>
</tr>
</tbody>
</table>

`datalad.api.create_sibling_webdav` function

`datalad.api.create_sibling_webdav(url, *, dataset=None, name=None, storage_name=None, mode='annex', credential=None, existing='error', recursive=False, recursion_limit=None)`

Create a sibling(-tandem) on a WebDAV server

WebDAV is a standard HTTP protocol extension for placing files on a server that is supported by a number of commercial storage services (e.g. 4shared.com, box.com), but also instances of cloud-storage solutions like Nextcloud or ownCloud. These software packages are also the basis for some institutional or public cloud storage solutions, such as EUDAT B2DROP.

For basic usage, only the URL with the desired dataset location on a WebDAV server needs to be specified for creating a sibling. However, the sibling setup can be flexibly customized (no storage sibling, or only a storage sibling, multi-version storage, or human-browsable single-version storage).

This command does not check for conflicting content on the WebDAV server!

When creating siblings recursively for a dataset hierarchy, subdataset exports are placed at their corresponding relative paths underneath the root location on the WebDAV server.

Collaboration on WebDAV siblings

The primary use case for WebDAV siblings is dataset deposition, where only one site is uploading dataset and file content updates. For collaborative workflows with multiple contributors, please make sure to consult the documentation on the underlying `datalad-annex`: Git remote helper for advice on appropriate setups: http://docs.datalad.org/projects/next/

Git-annex implementation details
Storage siblings are presently configured to NOT be enabled automatically on cloning a dataset. Due to a limitation of git-annex, this would initially fail (missing credentials). Instead, an explicit `datalad siblings enable --name <storage-sibling-name>` command must be executed after cloning. If necessary, it will prompt for credentials.

This command does not (and likely will not) support embedding credentials in the repository (see `embedcreds` option of the git-annex `webdav` special remote; https://git-annex.branchable.com/special_remotes/webdav), because such credential copies would need to be updated, whenever they change or expire. Instead, credentials are retrieved from DataLad’s credential system. In many cases, credentials are determined automatically, based on the HTTP authentication realm identified by a WebDAV server.

This command does not support setting up encrypted remotes (yet). Neither for the storage sibling, nor for the regular Git-remote. However, adding support for it is primarily a matter of extending the API of this command, and passing the respective options on to the underlying git-annex setup.

This command does not support setting up chunking for `webdav` storage siblings (https://git-annex.branchable.com/chunking).

### Examples

Create a WebDAV sibling tandem for storage of a dataset’s file content and revision history. A user will be prompted for any required credentials, if they are not yet known:

```bash
> create_sibling_webdav(url='https://webdav.example.com/myds')
```

Such a dataset can be cloned by DataLad via a specially crafted URL. Again, credentials are automatically determined, or a user is prompted to enter them:

```bash
> clone('datalad-annex::?type=webdav&encryption=none&url=https://webdav.example.com/myds')
```

A sibling can also be created with a human-readable file tree, suitable for data exchange with non-DataLad users, but only able to host a single version of each file:

```bash
> create_sibling_webdav(url='https://example.com/browseable', mode='filetree')
```

Cloning such dataset siblings is possible via a convenience URL:

```bash
> clone('webdavs://example.com/browseable')
```

In all cases, the storage sibling needs to explicitly enabled prior to file content retrieval:

```bash
> siblings('enable', name='example.com-storage')
```

### Parameters

- **url** -- URL identifying the sibling root on the target WebDAV server.
- **dataset** -- specify the dataset to process. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **name** -- name of the sibling. If none is given, the hostname-part of the WebDAV URL will be used. With `recursive`, the same name will be used to label all the subdatasets’ siblings. [Default: None]
- **storage_name** -- name of the storage sibling (git-annex special remote). Must not be identical to the sibling name. If not specified, defaults to the sibling name plus ‘-storage’ suffix.
If only a storage sibling is created, this setting is ignored, and the primary sibling name is used. [Default: None]

- **mode** -- Siblings can be created in various modes: full-featured sibling tandem, one for a dataset's Git history and one storage sibling to host any number of file versions ("annex"). A single sibling for the Git history only ("git-only"). A single annex sibling for multi-version file storage only ("annex-only"). As an alternative to the standard (annex) storage sibling setup that is capable of storing any number of historical file versions using a content hash layout ("annex"|"annex-only"), the 'filetree' mode can be used. This mode offers a human-readable data organization on the WebDAV remote that matches the file tree of a dataset (branch). However, it can, consequently, only store a single version of each file in the file tree. This mode is useful for depositing a single dataset snapshot for consumption without DataLad. The 'filetree' mode nevertheless allows for cloning such a single-version dataset, because the full dataset history can still be pushed to the WebDAV server. Git history hosting can also be turned off for this setup ('filetree- only'). When both a storage sibling and a regular sibling are created together, a publication dependency on the storage sibling is configured for the regular sibling in the local dataset clone. [Default: 'annex']

- **credential** -- name of the credential providing a user/password credential to be used for authorization. The credential can be supplied via configuration setting 'datalad.credential.<name>.user|secret', or environment variable DATA-LAD_CREDENTIAL_<NAME>_USER|SECRET, or will be queried from the active credential store using the provided name. If none is provided, the last-used credential for the authentication realm associated with the WebDAV URL will be used. Only if a credential name was given, it will be encoded in the URL of the created WebDAV Git remote, credential auto-discovery will be performed on each remote access. [Default: None]

- **existing** -- action to perform, if a (storage) sibling is already configured under the given name. In this case, sibling creation can be skipped ('skip') or the sibling (re-)configured ('reconfigure') in the dataset, or the command be instructed to fail ('error'). [Default: 'error']

- **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]

- **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; '<templates>' 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', 'repaths', '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']

datalad.api.credentials

datalad.api.credentials(action='query', spec=None, *, name=None, prompt=None, dataset=None)

Credential management and query

This command enables inspection and manipulation of credentials used throughout DataLad.

The command provides four basic actions:

**QUERY**

When executed without any property specification, all known credentials with all their properties will be yielded. Please note that this may not include credentials that only comprise of a secret and no other properties, or legacy credentials for which no trace in the configuration can be found. Therefore, the query results are not guaranteed to contain all credentials ever configured by DataLad.

When additional property/value pairs are specified, only credentials that have matching values for all given properties will be reported. This can be used, for example, to discover all suitable credentials for a specific "realm", if credentials were annotated with such information.

**SET**

This is the companion to 'get', and can be used to store properties and secret of a credential. Importantly, and in contrast to a 'get' operation, given properties with no values indicate a removal request. Any matching properties on record will be removed. If a credential is to be stored for which no secret is on record yet, an interactive session will prompt a user for a manual secret entry.

Only changed properties will be contained in the result record.

The appearance of the interactive secret entry can be configured with the two settings `datalad.credentials.repeat-secret-entry` and `datalad.credentials.hidden-secret-entry`.

**REMOVE**

This action will remove any secret and properties associated with a credential identified by its name.

**GET** (plumbing operation)

This is a read-only action that will never store (updates of) credential properties or secrets. Given properties will amend/overwrite those already on record. When properties with no value are given, and also no value for the respective properties is on record yet, their value will be requested interactively, if a prompt text was provided too. This can be used to ensure a complete credential record, comprising any number of properties.

Details on credentials

Chapter 2. API
A credential comprises any number of properties, plus exactly one secret. There are no constraints on the format or property values or the secret, as long as they are encoded as a string.

Credential properties are normally stored as configuration settings in a user's configuration ('global' scope) using the naming scheme:

\[ \text{datalad.credential.<name>.<property>} \]

Therefore both credential name and credential property name must be syntax-compliant with Git configuration items. For property names this means only alphanumeric characters and dashes. For credential names virtually no naming restrictions exist (only null-byte and newline are forbidden). However, when naming credentials it is recommended to use simple names in order to enable convenient one-off credential overrides by specifying DataLad configuration items via their environment variable counterparts (see the documentation of the \text{configuration} command for details. In short, avoid underscores and special characters other than \''\text{'} and '-'\'.

While there are no constraints on the number and nature of credential properties, a few particular properties are recognized on used for particular purposes:

- \'secret\': always refers to the single secret of a credential
- \'type\': identifies the type of a credential. With each standard type, a list of mandatory properties is associated (see below)
- \'last-used\': is an ISO 8601 format time stamp that indicated the last (successful) usage of a credential

Standard credential types and properties

The following standard credential types are recognized, and their mandatory field with their standard names will be automatically included in a 'get' report.

- \'user_password\': with properties \'user\', and the password as secret
- \'token\': only comprising the token as secret
- \'aws-s3\': with properties \'key-id\', \'session\', \'expiration\', and the secret_id as the credential secret

Legacy support

DataLad credentials not configured via this command may not be fully discoverable (i.e., including all their properties). Discovery of such legacy credentials can be assisted by specifying a dedicated \'type\' property.

Examples

Report all discoverable credentials:

\[
> \text{credentials()} \]

Set a new credential mycred & input its secret interactively:

\[
> \text{credentials(\'set\', name=\'mycred\')} \]

Remove a credential's type property:

\[
> \text{credentials(\'set\', name=\'mycred\', spec=\{'type\': \text{None}\})} \]

Get all information on a specific credential in a structured record:

\[
> \text{credentials(\'get\', name=\'mycred\')} \]

Upgrade a legacy credential by annotating it with a \'type\' property:
Set a new credential of type `user_password`, with a given user property, and input its secret interactively:

```python
> credentials('set', name='legacycred', spec={'type': 'user_password'})
```

Obtain a (possibly yet undefined) credential with a minimum set of properties. All missing properties and secret will be prompted for, no information will be stored! This is mostly useful for ensuring availability of an appropriate credential in an application context:

```python
> credentials('set', name='mycred', spec={'type': 'user_password', 'user': 'admin', 'newproperty': 'None'})
```

Obtain a (possibly yet undefined) credential with a minimum set of properties. All missing properties and secret will be prompted for, no information will be stored! This is mostly useful for ensuring availability of an appropriate credential in an application context:

```python
> credentials('get', prompt='Can I haz info plz?', name='newcred', spec={
    'type': 'user_password',
    'user': 'admin',
    'newproperty': None
})
```

### Parameters

- **action** -- which action to perform. [Default: `query`]
- **spec** -- specification of credential properties. Properties are given as name/value pairs. Properties with a `None` value indicate a property to be deleted (action `set`), or a property to be entered interactively, when no value is set yet, and a prompt text is given (action `get`). All property names are case-insensitive, must start with a letter or a digit, and may only contain `-' apart from these characters. Property specifications should be given as a dictionary, e.g., `spec={'type': 'user_password'}`. However, a CLI-like list of string arguments is also supported, e.g., `spec=[type=user_password]`. [Default: None]
- **name** -- name of a credential to set, get, or remove. [Default: None]
- **prompt** -- message to display when entry of missing credential properties is required for action `get`. This can be used to present information on the nature of a credential and for instructions on how to obtain a credential. [Default: None]
- **dataset** -- specify a dataset whose configuration to inspect rather than the global (user) settings. [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 `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; `<templates>` 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', 'repaths', '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']

datalad.api.download
datalad.api.download(spec, *, dataset=None, force=None, credential=None, hash=None)

Download from URLs

This command is the front-end to an extensible framework for performing downloads from a variety of URL schemes. Built-in support for the schemes 'http', 'https', 'file', and 'ssh' is provided. Extension packages may add additional support.

In contrast to other downloader tools, this command integrates with the DataLad credential management and is able to auto-discover credentials. If no credential is available, it automatically prompts for them, and offers to store them for re-use after a successful authentication.

Simultaneous hashing (checksumming) of downloaded content is supported with user-specified algorithms.

The command can process any number of downloads (serially). it can read download specifications from (command line) arguments, files, or STDIN. It can deposit downloads to individual files, or stream to STDOUT.

Implementation and extensibility

Each URL scheme is processed by a dedicated handler. Additional schemes can be supported by sub-classing datalad.next.url_operations.UrlOperations and implementing the download() method. Extension packages can register new handlers, by patching them into the datalad.next.download._urlscheme_handlers registry dict.

**Examples**

Download webpage to "myfile.txt":

```
> download("http://example.com": "myfile.txt")
```

Read download specification from STDIN (e.g. JSON-lines):

```
> download("-")
```

Simultaneously hash download, hexdigest reported in result record:

```
> download("http://example.com/data.xml", hash=["sha256"])
```

Download from SSH server:

2.1. High-level API commands
Parameters

- **spec** -- Download sources and targets can be given in a variety of formats: as a URL, or as a URL-path-pair that is mapping a source URL to a dedicated download target path. Any number of URLs or URL-path-pairs can be provided, either as an argument list, or read from a file (one item per line). Such a specification input file can be given as a path to an existing file (as a single value, not as part of a URL-path-pair). When the special path identifier ‘~’ is used, the download is written to STDOUT. A specification can also be read in JSON-lines encoding (each line being a string with a URL or an object mapping a URL-string to a path-string). In addition, specifications can also be given as a list or URLs, or as a list of dicts with a URL to path mapping. Paths are supported in string form, or as Path objects.

- **dataset** -- Dataset to be used as a configuration source. Beyond reading configuration items, this command does not interact with the dataset. [Default: None]

- **force** -- By default, a target path for a download must not exist yet. 'force-overwrite' disabled this check. [Default: None]

- **credential** -- name of a credential to be used for authorization. If no credential is identified, the last-used credential for the authentication realm associated with the download target will be used. If there is no credential available yet, it will be prompted for. Once used successfully, a prompt for entering to save such a new credential will be presented. [Default: None]

- **hash** -- Name of a hashing algorithm supported by the Python 'hashlib' module, e.g. 'md5' or 'sha256'. [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', 'repaths', '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']

datalad.api.ls_file_collection

datalad.api.ls_file_collection(type: str, collection: CollectionSpec, *, hash: str | List[str] | None = None)

Report information on files in a collection

This is a utility that can be used to query information on files in different file collections. The type of information reported varies across collection types. However, each result at minimum contains some kind of identifier for the collection ('collection' property), and an identifier for the respective collection item ('item' property). Each result also contains a type property that indicates particular type of file that is being reported on. In most cases this will be file, but other categories like symlink or directory are recognized too.

If a collection type provides file-access, this command can compute one or more hashes (checksums) for any file in a collection.

Supported file collection types are:

directory
Reports on the content of a given directory (non-recursively). The collection identifier is the path of the directory. Item identifiers are the name of a file within that directory. Standard properties like size, mtime, or link_target are included in the report. When hashes are computed, an fp property with a file-like is provided. Reading file data from it requires a seek(0) in most cases. This file handle is only open when items are yielded directly by this command (return_type='generator') and only until the next result is yielded.

tarfile
Reports on members of a TAR archive. The collection identifier is the path of the TAR file. Item identifiers are the relative paths of archive members within the archive. Reported properties are similar to the directory collection type. When hashes are computed, an fp property with a file-like is provided. Reading file data from it requires a seek(0) in most cases. This file handle is only open when items are yielded directly by this command (return_type='generator') and only until the next result is yielded.

Examples

Report on the content of a directory:

```python
> records = ls_file_collection("directory", "/tmp")
```

Report on the content of a TAR archive with MD5 and SHA1 file hashes:

```python
> records = ls_file_collection("tarfile", "myarchive.tar.gz", hash=["md5", "sha1"])
```

Parameters

• type -- Name of the type of file collection to report on.
• **collection** -- identifier or location of the file collection to report on. Depending on the type of collection to process, the specific nature of this parameter can be different. A common identifier for a file collection is a path (to a directory, to an archive), but might also be a URL. See the documentation for details on supported collection types.

• **hash** -- One or more names of algorithms to be used for reporting file hashes. They must be supported by the Python 'hashlib' module, e.g. 'md5' or 'sha256'. Reporting file hashes typically implies retrieving/reading file content. This processing may also enable reporting of additional properties that may otherwise not be readily available. [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']
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

**datalad.api.tree**

`datalad.api.tree(path='.', *, depth=None, recursive=False, recursion_limit=None, include_files=False, include_hidden=False)`

Visualize directory and dataset hierarchies

This command mimics the UNIX/MS-DOS 'tree' utility to generate and display a directory tree, with DataLad-specific enhancements.

It can serve the following purposes:

1. Glorified 'tree' command
2. Dataset discovery
3. Programmatic directory traversal

**Glorified 'tree' command**

The rendered command output uses 'tree'-style visualization:

```
/tmp/mydir
    [DS~0] ds_A/
    [DS~1] subds_A/
    [DS~0] ds_B/
        dir_B/
            file.txt
            subdir_B/
                [DS~1] subds_B0/
                    [DS~1] (not installed) subds_B1/
```

5 datasets, 2 directories, 1 file

Dataset paths are prefixed by a marker indicating subdataset hierarchy level, like [DS~1]. This is the absolute subdataset level, meaning it may also take into account superdatasets located above the tree root and thus not included in the output. If a subdataset is registered but not installed (such as after a non-recursive `datalad clone`), it will be prefixed by (not installed). Only DataLad datasets are considered, not pure git/git-annex repositories.

The 'report line' at the bottom of the output shows the count of displayed datasets, in addition to the count of directories and files. In this context, datasets and directories are mutually exclusive categories.

By default, only directories (no files) are included in the tree, and hidden directories are skipped. Both behaviours can be changed using command options.

Symbolic links are always followed. This means that a symlink pointing to a directory is traversed and counted as a directory (unless it potentially creates a loop in the tree).

**Dataset discovery**

Using the recursive or recursion_limit option, this command generates the layout of dataset hierarchies based on subdataset nesting level, regardless of their location in the filesystem.

In this case, tree depth is determined by subdataset depth. This mode is thus suited for discovering available datasets when their location is not known in advance.

By default, only datasets are listed, without their contents. If depth is specified additionally, the contents of each dataset will be included up to depth directory levels (excluding subdirectories that are themselves datasets).

Tree filtering options such as include_hidden only affect which directories are reported as dataset contents, not which directories are traversed to find datasets.
Performance note: since no assumption is made on the location of datasets, running this command with the recursive or recursion_limit option does a full scan of the whole directory tree. As such, it can be significantly slower than a call with an equivalent output that uses depth to limit the tree instead.

Programmatic directory traversal

The command yields a result record for each tree node (dataset, directory or file). The following properties are reported, where available:

"path"
Absolute path of the tree node

"type"
Type of tree node: "dataset", "directory" or "file"

"depth"
Directory depth of node relative to the tree root

"exhausted_levels"
Depth levels for which no nodes are left to be generated (the respective subtrees have been 'exhausted')

"count"
Dict with cumulative counts of datasets, directories and files in the tree up until the current node. File count is only included if the command is run with the include_files option.

"dataset_depth"
Subdataset depth level relative to the tree root. Only included for node type "dataset".

"dataset_abs_depth"
Absolute subdataset depth level. Only included for node type "dataset".

"dataset_is_installed"
Whether the registered subdataset is installed. Only included for node type "dataset".

"symlink_target"
If the tree node is a symlink, the path to the link target

"is_broken_symlink"
If the tree node is a symlink, whether it is a broken symlink

Examples

Show up to 3 levels of subdirectories below the current directory, including files and hidden contents:

```
> tree(depth=3, include_files=True, include_hidden=True)
```

Find all top-level datasets located anywhere under /tmp:

```
> tree('/tmp', recursion_limit=0)
```

Report all subdatasets recursively and their directory contents, up to 1 subdirectory deep within each dataset:

```
> tree(recursive=True, depth=1)
```

Parameters

- path -- path to directory from which to generate the tree. Defaults to the current directory.
  [Default: '.']
• **depth** -- limit the tree to maximum level of subdirectories. If not specified, will generate the full tree with no depth constraint. If paired with **recursive** or **recursion_limit**, refers to the maximum directory level to output below each dataset. [Default: None]

• **recursive** (*bool, optional*) -- produce a dataset tree of the full hierarchy of nested subdatasets. *Note:* may have slow performance on large directory trees. [Default: False]

• **recursion_limit** -- limit the dataset tree to maximum level of nested subdatasets. 0 means include only top-level datasets, 1 means top-level datasets and their immediate subdatasets, etc. *Note:* may have slow performance on large directory trees. [Default: None]

• **include_files** (*bool, optional*) -- include files in the tree. [Default: False]

• **include_hidden** (*bool, optional*) -- include hidden files/directories in the tree. This option does not affect which directories will be searched for datasets when specifying **recursive** or **recursion_limit**. For example, datasets located underneath the hidden folder .datalad will be reported even if **include_hidden** is omitted. [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 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']
2.2 Command line reference

2.2.1 Command line reference

datalad create-sibling-webdav

Synopsis


Description

Create a sibling(-tandem) on a WebDAV server

WebDAV is a standard HTTP protocol extension for placing files on a server that is supported by a number of commercial storage services (e.g. 4shared.com, box.com), but also instances of cloud-storage solutions like Nextcloud or ownCloud. These software packages are also the basis for some institutional or public cloud storage solutions, such as EUDAT B2DROP.

For basic usage, only the URL with the desired dataset location on a WebDAV server needs to be specified for creating a sibling. However, the sibling setup can be flexibly customized (no storage sibling, or only a storage sibling, multiversion storage, or human-browsable single-version storage).

This command does not check for conflicting content on the WebDAV server!

When creating siblings recursively for a dataset hierarchy, subdataset exports are placed at their corresponding relative paths underneath the root location on the WebDAV server.

Collaboration on WebDAV siblings

The primary use case for WebDAV siblings is dataset deposition, where only one site is uploading dataset and file content updates. For collaborative workflows with multiple contributors, please make sure to consult the documentation on the underlying datalad-annex:: Git remote helper for advice on appropriate setups: http://docs.datalad.org/projects/next/

Git-annex implementation details

Storage siblings are presently configured to NOT be enabled automatically on cloning a dataset. Due to a limitation of git-annex, this would initially fail (missing credentials). Instead, an explicit datalad siblings enable --name <storage-sibling-name> command must be executed after cloning. If necessary, it will prompt for credentials.

This command does not (and likely will not) support embedding credentials in the repository (see embedcreds option of the git-annex webdav special remote; https://git-annex.branchable.com/special_remotes/webdav), because such credential copies would need to be updated, whenever they change or expire. Instead, credentials are retrieved from DataLad’s credential system. In many cases, credentials are determined automatically, based on the HTTP authentication realm identified by a WebDAV server.

This command does not support setting up encrypted remotes (yet). Neither for the storage sibling, nor for the regular Git-remote. However, adding support for it is primarily a matter of extending the API of this command, and passing the respective options on to the underlying git-annex setup.

This command does not support setting up chunking for webdav storage siblings (https://git-annex.branchable.com/chunking).

Examples
Create a WebDAV sibling tandem for storage of a dataset's file content and revision history. A user will be prompted for any required credentials, if they are not yet known:

```shell
% datalad create-sibling-webdav "https://webdav.example.com/myds"
```

Such a dataset can be cloned by DataLad via a specially crafted URL. Again, credentials are automatically determined, or a user is prompted to enter them:

```shell
% datalad clone "datalad-annex::?type=webdav&encryption=none&url=https://webdav.example.com/myds"
```

A sibling can also be created with a human-readable file tree, suitable for data exchange with non-DataLad users, but only able to host a single version of each file:

```shell
% datalad create-sibling-webdav --mode filetree "https://example.com/browsable"
```

Cloning such dataset siblings is possible via a convenience URL:

```shell
% datalad clone "webdavs://example.com/browsable"
```

In all cases, the storage sibling needs to explicitly enabled prior to file content retrieval:

```shell
% datalad siblings enable --name example.com-storage
```

### Options

**URL**

URL identifying the sibling root on the target WebDAV server.

- `-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 process. If no dataset is given, an attempt is made to identify the dataset based on the current working directory.

- `-s NAME, --name NAME`

name of the sibling. If none is given, the hostname-part of the WebDAV URL will be used. With RECURSIVE, the same name will be used to label all the subdatasets' siblings.
--storage-name NAME

name of the storage sibling (git-annex special remote). Must not be identical to the sibling name. If not specified, defaults to the sibling name plus '-storage' suffix. If only a storage sibling is created, this setting is ignored, and the primary sibling name is used.

--mode MODE

Siblings can be created in various modes: full-featured sibling tandem, one for a dataset's Git history and one storage sibling to host any number of file versions ('annex'). A single sibling for the Git history only ('git-only'). A single annex sibling for multi-version file storage only ('annex-only'). As an alternative to the standard (annex) storage sibling setup that is capable of storing any number of historical file versions using a content hash layout ('annex'[annex-only]'), the 'filetree' mode can be used. This mode offers a human-readable data organization on the WebDAV remote that matches the file tree of a dataset (branch). However, it can, consequently, only store a single version of each file in the file tree. This mode is useful for depositing a single dataset snapshot for consumption without DataLad. The 'filetree' mode nevertheless allows for cloning such a single-version dataset, because the full dataset history can still be pushed to the WebDAV server. Git history hosting can also be turned off for this setup ('filetree-only'). When both a storage sibling and a regular sibling are created together, a publication dependency on the storage sibling is configured for the regular sibling in the local dataset clone. [Default: 'annex']

--credential NAME

name of the credential providing a user/password credential to be used for authorization. The credential can be supplied via configuration setting 'datalad.credential.<name>.user|secret', or environment variable DATALAD_CREDENTIAL_<NAME>_USER|SECRET, or will be queried from the active credential store using the provided name. If none is provided, the last-used credential for the authentication realm associated with the WebDAV URL will be used. Only if a credential name was given, it will be encoded in the URL of the created WebDAV Git remote, credential auto-discovery will be performed on each remote access.

--existing EXISTING

action to perform, if a (storage) sibling is already configured under the given name. In this case, sibling creation can be skipped ('skip') or the sibling (re-)configured ('reconfigure') in the dataset, or the command be instructed to fail ('error'). [Default: 'error']

-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
--version

show the module and its version which provides the command

Authors

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

datalad credentials

Synopsis

datalad credentials [-h] [--prompt PROMPT] [-d DATASET] [--version] [action] [[name].
˓→[:]property=[value] ...]

Description

Credential management and query

This command enables inspection and manipulation of credentials used throughout DataLad.

The command provides four basic actions:

QUERY

When executed without any property specification, all known credentials with all their properties will be yielded. Please note that this may not include credentials that only comprise of a secret and no other properties, or legacy credentials for which no trace in the configuration can be found. Therefore, the query results are not guaranteed to contain all credentials ever configured by DataLad.

When additional property/value pairs are specified, only credentials that have matching values for all given properties will be reported. This can be used, for example, to discover all suitable credentials for a specific "realm", if credentials were annotated with such information.

SET

This is the companion to 'get', and can be used to store properties and secret of a credential. Importantly, and in contrast to a 'get' operation, given properties with no values indicate a removal request. Any matching properties on record will be removed. If a credential is to be stored for which no secret is on record yet, an interactive session will prompt a user for a manual secret entry.

Only changed properties will be contained in the result record.

The appearance of the interactive secret entry can be configured with the two settings datalad.credentials.repeat-secret-entry and datalad.credentials.hidden-secret-entry.

REMOVE

This action will remove any secret and properties associated with a credential identified by its name.

GET (plumbing operation)

This is a read-only action that will never store (updates of) credential properties or secrets. Given properties will amend/overwrite those already on record. When properties with no value are given, and also no value for the respective properties is on record yet, their value will be requested interactively, if a --prompt text was provided too. This can be used to ensure a complete credential record, comprising any number of properties.
Details on credentials

A credential comprises any number of properties, plus exactly one secret. There are no constraints on the format or
property values or the secret, as long as they are encoded as a string.

Credential properties are normally stored as configuration settings in a user's configuration ('global' scope) using the
naming scheme:

    datalad.credential.<name>.<property>

Therefore both credential name and credential property name must be syntax-compliant with Git configuration items.
For property names this means only alphanumeric characters and dashes. For credential names virtually no naming
restrictions exist (only null-byte and newline are forbidden). However, when naming credentials it is recommended to
use simple names in order to enable convenient one-off credential overrides by specifying DataLad configuration items
via their environment variable counterparts (see the documentation of the configuration command for details. In
short, avoid underscores and special characters other than ',', and '.'.

While there are no constraints on the number and nature of credential properties, a few particular properties are recog-
nized on used for particular purposes:

- 'secret': always refers to the single secret of a credential
- 'type': identifies the type of a credential. With each standard type, a list of mandatory properties is associated
  (see below)
- 'last-used': is an ISO 8601 format time stamp that indicated the last (successful) usage of a credential

Standard credential types and properties

The following standard credential types are recognized, and their mandatory field with their standard names will be
automatically included in a 'get' report.

- 'user_password': with properties 'user', and the password as secret
- 'token': only comprising the token as secret
- 'aws-s3': with properties 'key-id', 'session', 'expiration', and the secret_id as the credential secret

Legacy support

DataLad credentials not configured via this command may not be fully discoverable (i.e., including all their properties).
Discovery of such legacy credentials can be assisted by specifying a dedicated 'type' property.

Examples

Report all discoverable credentials:

    % datalad credentials

Set a new credential mycred & input its secret interactively:

    % datalad credentials set mycred

Remove a credential's type property:

    % datalad credentials set mycred :type

Get all information on a specific credential in a structured record:

    % datalad -f json credentials get mycred

Upgrade a legacy credential by annotating it with a 'type' property:
% datalad credentials set legacycred type=user_password

Set a new credential of type user_password, with a given user property, and input its secret interactively:

% datalad credentials set mycred type=user_password user=<username>

Obtain a (possibly yet undefined) credential with a minimum set of properties. All missing properties and secret will be prompted for, no information will be stored! This is mostly useful for ensuring availability of an appropriate credential in an application context:

% datalad credentials --prompt 'can I haz info plz?' get newcred :newproperty

Options

action

which action to perform. [Default: 'query']

[name] [:]property[=value]

specification of a credential name and credential properties. Properties are either given as name/value pairs or as a property name prefixed by a colon. Properties prefixed with a colon indicate a property to be deleted (action 'set'), or a property to be entered interactively, when no value is set yet, and a prompt text is given (action 'get'). All property names are case-insensitive, must start with a letter or a digit, and may only contain '-' apart from these characters.

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

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

--prompt PROMPT

message to display when entry of missing credential properties is required for action 'get'. This can be used to present information on the nature of a credential and for instructions on how to obtain a credential.

-d DATASET, --dataset DATASET

specify a dataset whose configuration to inspect rather than the global (user) settings.
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

--version

show the module and its version which provides the command

Authors

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

datalad download

Synopsis

```bash
datalad download [-h] [-d DATASET] [--force {overwrite-existing}] [--credential NAME] [--hash ALGORITHM] [--version] <path>|<url>|<url-path-pair> [<path>|<url>|<url-path-pair>...
```

Description

Download from URLs

This command is the front-end to an extensible framework for performing downloads from a variety of URL schemes. Built-in support for the schemes 'http', 'https', 'file', and 'ssh' is provided. Extension packages may add additional support.

In contrast to other downloader tools, this command integrates with the DataLad credential management and is able to auto-discover credentials. If no credential is available, it automatically prompts for them, and offers to store them for re-use after a successful authentication.

Simultaneous hashing (checksumming) of downloaded content is supported with user-specified algorithms.

The command can process any number of downloads (serially). it can read download specifications from (command line) arguments, files, or STDIN. It can deposit downloads to individual files, or stream to STDOUT.

Implementation and extensibility

Each URL scheme is processed by a dedicated handler. Additional schemes can be supported by sub-classing datalad_next.url_operations.UrlOperations and implementing the `download()` method. Extension packages can register new handlers, by patching them into the `datalad_next.download._urlscheme_handlers` registry dict.

Examples

Download webpage to "myfile.txt":

```bash
% datalad download "http://example.com myfile.txt"
```

Read download specification from STDIN (e.g. JSON-lines):

```bash
% datalad download -
```

Simultaneously hash download, hexdigest reported in result record:

```bash
% datalad download --hash sha256 http://example.com/data.xml"
```

Download from SSH server:
Stream a download to STDOUT:

```bash
% datalad -f disabled download "http://example.com -"
```

## Options

### `<path>|<url>|<url-path-pair>`

Download sources and targets can be given in a variety of formats: as a URL, or as a URL-path-pair that is mapping a source URL to a dedicated download target path. Any number of URLs or URL-path-pairs can be provided, either as an argument list, or read from a file (one item per line). Such a specification input file can be given as a path to an existing file (as a single value, not as part of a URL-path-pair). When the special path identifier `'-'` is used, the download is written to STDOUT. A specification can also be read in JSON-lines encoding (each line being a string with a URL or an object mapping a URL-string to a path-string).

### `-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 be used as a configuration source. Beyond reading configuration items, this command does not interact with the dataset.

### `--force {overwrite-existing}`

By default, a target path for a download must not exist yet. `force-overwrite` disabled this check.

### `--credential NAME`

Name of a credential to be used for authorization. If no credential is identified, the last-used credential for the authentication realm associated with the download target will be used. If there is no credential available yet, it will be prompted for. Once used successfully, a prompt for entering to save such a new credential will be presented.

### `--hash ALGORITHM`

Name of a hashing algorithm supported by the Python `hashlib` module, e.g. `md5` or `sha256`. This option can be given more than once.

---

**2.2. Command line reference**
--version

show the module and its version which provides the command

Authors

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

datalad ls-file-collection

Synopsis

datalad ls-file-collection [-h] [--hash ALGORITHM] [--version] {directory,tarfile,
gitworktree} ID/LOCATION

Description

Report information on files in a collection

This is a utility that can be used to query information on files in different file collections. The type of information reported varies across collection types. However, each result at minimum contains some kind of identifier for the collection (‘collection' property), and an identifier for the respective collection item (‘item' property). Each result also contains a type property that indicates particular type of file that is being reported on. In most cases this will be file, but other categories like symlink or directory are recognized too.

If a collection type provides file-access, this command can compute one or more hashes (checksums) for any file in a collection.

Supported file collection types are:

directory
Reports on the content of a given directory (non-recursively). The collection identifier is the path of the directory. Item identifiers are the name of a file within that directory. Standard properties like size, mtime, or link_target are included in the report.

tarfile
Reports on members of a TAR archive. The collection identifier is the path of the TAR file. Item identifiers are the relative paths of archive members within the archive. Reported properties are similar to the directory collection type.

Examples

Report on the content of a directory:

```
% datalad -f json ls-file-collection directory /tmp
```

Report on the content of a TAR archive with MD5 and SHA1 file hashes:

```
% datalad -f json ls-file-collection --hash md5 --hash sha1 tarfile myarchive.tar.gz
```

Register URLs for files in a directory that is also reachable via HTTP. This uses ls-file-collection for listing files and computing MD5 hashes, then using jq to filter and transform the output (just file records, and in a JSON array), and passes them to addurls, which generates annex keys/files and assigns URLs. When the command finishes, the dataset contains no data, but can retrieve the files after confirming their availability (i.e., via git annex fsck):
% datalad -f json ls-file-collection directory wwwdir --hash md5 \  | jq '. | select(.type == "file")' \  | jq --slurp . \  | datalad addurls --key 'et:MD5-s{size}--{hash-md5}' - 'https://example.com/{item}'

Options

{directory,tarfile,gitworktree}

Name of the type of file collection to report on.

**ID/LOCATION**

identifier or location of the file collection to report on. Depending on the type of collection to process, the specific nature of this parameter can be different. A common identifier for a file collection is a path (to a directory, to an archive), but might also be a URL. See the documentation for details on supported collection types.

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

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

- **--hash ALGORITHM**

One or more names of algorithms to be used for reporting file hashes. They must be supported by the Python 'hashlib' module, e.g. 'md5' or 'sha256'. Reporting file hashes typically implies retrieving/reading file content. This processing may also enable reporting of additional properties that may otherwise not be readily available. This option can be given more than once.

- **--version**

show the module and its version which provides the command

**Authors**

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

datalad tree

**Synopsis**

datalad tree [-h] [-L DEPTH] [-r] [-R LEVELS] [--include-files] [--include-hidden] [--version] [path]
Description

Visualize directory and dataset hierarchies

This command mimics the UNIX/MS-DOS 'tree' utility to generate and display a directory tree, with DataLad-specific enhancements.

It can serve the following purposes:

1. Glorified 'tree' command
2. Dataset discovery
3. Programmatic directory traversal

Glorified 'tree' command

The rendered command output uses 'tree'-style visualization:

```
/tmp/mydir
    [DS~0] ds_A/
    [DS~1] subds_A/
    [DS~0] ds_B/
       dir_B/
          file.txt
          subdir_B/
          [DS~1] subds_B0/
          [DS~1] (not installed) subds_B1/
```

5 datasets, 2 directories, 1 file

Dataset paths are prefixed by a marker indicating subdataset hierarchy level, like [DS~1]. This is the absolute subdataset level, meaning it may also take into account superdatasets located above the tree root and thus not included in the output. If a subdataset is registered but not installed (such as after a non-recursive datalad clone), it will be prefixed by (not installed). Only DataLad datasets are considered, not pure git/git-annex repositories.

The 'report line' at the bottom of the output shows the count of displayed datasets, in addition to the count of directories and files. In this context, datasets and directories are mutually exclusive categories.

By default, only directories (no files) are included in the tree, and hidden directories are skipped. Both behaviours can be changed using command options.

Symbolic links are always followed. This means that a symlink pointing to a directory is traversed and counted as a directory (unless it potentially creates a loop in the tree).

Dataset discovery

Using the --recursive or --recursion-limit option, this command generates the layout of dataset hierarchies based on subdataset nesting level, regardless of their location in the filesystem.

In this case, tree depth is determined by subdataset depth. This mode is thus suited for discovering available datasets when their location is not known in advance.

By default, only datasets are listed, without their contents. If --depth is specified additionally, the contents of each dataset will be included up to --depth directory levels (excluding subdirectories that are themselves datasets).

Tree filtering options such as --include-hidden only affect which directories are reported as dataset contents, not which directories are traversed to find datasets.

Performance note: since no assumption is made on the location of datasets, running this command with the --recursive or --recursion-limit option does a full scan of the whole directory tree. As such, it can be significantly slower than a call with an equivalent output that uses --depth to limit the tree instead.
Programmatic directory traversal

The command yields a result record for each tree node (dataset, directory or file). The following properties are reported, where available:

"path"
   Absolute path of the tree node

"type"
   Type of tree node: "dataset", "directory" or "file"

"depth"
   Directory depth of node relative to the tree root

"exhausted_levels"
   Depth levels for which no nodes are left to be generated (the respective subtrees have been 'exhausted')

"count"
   Dict with cumulative counts of datasets, directories and files in the tree up until the current node. File count is only included if the command is run with the --include-files option.

"dataset_depth"
   Subdataset depth level relative to the tree root. Only included for node type "dataset".

"dataset_abs_depth"
   Absolute subdataset depth level. Only included for node type "dataset".

"dataset_is_installed"
   Whether the registered subdataset is installed. Only included for node type "dataset".

"symlink_target"
   If the tree node is a symlink, the path to the link target

"is_broken_symlink"
   If the tree node is a symlink, whether it is a broken symlink

Examples

Show up to 3 levels of subdirectories below the current directory, including files and hidden contents:

% datalad tree -L 3 --include-files --include-hidden

Find all top-level datasets located anywhere under /tmp:

% datalad tree /tmp -R 0

Report all subdatasets recursively and their directory contents, up to 1 subdirectory deep within each dataset:

% datalad tree -r -L 1

Options

path

path to directory from which to generate the tree. Defaults to the current directory. [Default: '.']
-h, --help, --help-np

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

-L DEPTH, --depth DEPTH

limit the tree to maximum level of subdirectories. If not specified, will generate the full tree with no depth constraint. If paired with --recursive or --recursion-limit, refers to the maximum directory level to output below each dataset.

-r, --recursive

produce a dataset tree of the full hierarchy of nested subdatasets. Note: may have slow performance on large directory trees.

-R LEVELS, --recursion-limit LEVELS

limit the dataset tree to maximum level of nested subdatasets. 0 means include only top-level datasets, 1 means top-level datasets and their immediate subdatasets, etc. Note: may have slow performance on large directory trees.

--include-files

include files in the tree.

--include-hidden

include hidden files/directories in the tree. This option does not affect which directories will be searched for datasets when specifying --recursive or --recursion-limit. For example, datasets located underneath the hidden folder .datalad will be reported even if --include-hidden is omitted.

--version

show the module and its version which provides the command.

Authors

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

datalad-next comprises a number of more-or-less self-contained mini-packages providing particular functionality.

2.3.1 Python tooling

datalad-next comprises a number of more-or-less self-contained mini-packages providing particular functionality. These implementations are candidates for a migration into the DataLad core package, and are provided here for immediate use. If and when components are migrated, transition modules will be kept to prevent API breakage in dependent packages.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>archive_operations</td>
<td>Handler for operations on various archive types</td>
</tr>
<tr>
<td>commands</td>
<td>Essential tooling for implementing DataLad commands</td>
</tr>
<tr>
<td>config</td>
<td>Configuration query and manipulation</td>
</tr>
<tr>
<td>constraints</td>
<td>Data validation, coercion, and parameter documentation</td>
</tr>
<tr>
<td>credman</td>
<td>Credential management</td>
</tr>
<tr>
<td>datasets</td>
<td>Representations of DataLad datasets built on git/git-annex repositories</td>
</tr>
<tr>
<td>exceptions</td>
<td>All custom exceptions used in datalad-next</td>
</tr>
<tr>
<td>iter_collections</td>
<td>Iterators for particular types of collections</td>
</tr>
<tr>
<td>runners</td>
<td>Execution of subprocesses</td>
</tr>
<tr>
<td>tests.fixtures</td>
<td>Collection of fixtures for facilitation test implementations</td>
</tr>
<tr>
<td>types</td>
<td>Custom types and dataclasses</td>
</tr>
<tr>
<td>uis</td>
<td>UI abstractions for user communication</td>
</tr>
<tr>
<td>url_operations</td>
<td>Handlers for operations on various URL types and protocols</td>
</tr>
<tr>
<td>utils</td>
<td>Assorted utility functions</td>
</tr>
</tbody>
</table>

**datalad_next.archive_operations**

Handler for operations on various archive types

All handlers implement the API defined by *ArchiveOperations*.

Available handlers:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tarfile</td>
<td>TAR archive operation handler</td>
</tr>
</tbody>
</table>

**datalad_next.archive_operations.tarfile**

TAR archive operation handler

```python
class datalad_next.archive_operations.tarfile.TarArchiveOperations(location: Path, *, cfg: ConfigManager | None = None)
```

Bases: *ArchiveOperations*

Handler for a TAR archive on a local file system
Any methods that take an archive item/member name as an argument accept a POSIX path string, or any PurePath instance.

```
close() → None
```
Closes any opened TAR file handler

```
open(item: str | PurePosixPath) → Generator[IO | None]
```
Get a file-like for a TAR archive item

The file-like object allows to read from the archive-item specified by `item`.

**Parameters**

- `item` (str | PurePath) -- The identifier must be a POSIX path string, or a PurePath instance.

**Returns**

A file-like object to read bytes from the item, if the item is a regular file, else `None`. (This is returned by the context manager that is created via the decorator `@contextmanager`.)

**Return type**

IO | None

**Raises**

- `KeyError` -- If no item with the name `item` can be found in the tar-archive

```
property tarfile: TarFile
```
Returns TarFile instance, after creating it on-demand

The instance is cached, and needs to be released by calling `.close()` if called outside a context manager.

```
class datalad_next.archive_operations.ArchiveOperations(location: Any, *, cfg: ConfigManager | None = None)
```
Base class of all archives handlers

Any handler can be used as a context manager to adequately acquire and release any resources necessary to access an archive. Alternatively, the `close()` method can be called, when archive access is no longer needed.

In addition to the `open()` method for accessing archive item content, each handler implements the standard `__contains__()` and `__iter__()`.

`__contains__()` reports whether the archive contains an items of a given identifier.

`__iter__()` provides an iterator that yields `FileSystemItem` instances with information on each archive item.

```
property cfg: ConfigManager
```
ConfigManager given to the constructor, or the session default

```
close() → None
```
Default implementation for closing a archive handler

This default implementation does nothing.

```
abstract open(item: Any) → Generator[IO | None]
```
Get a file-like for an archive item

**Parameters**

- `item` -- Any identifier for an archive item supported by a particular handler
**datalad_next.commands**

Essential tooling for implementing DataLad commands

This module provides the advanced command base class `ValidatedInterface`, for implementing commands with uniform argument validation and structured error reporting.

Beyond that, any further components necessary to implement command are imported in this module to offer a one-stop-shop experience. This includes `build_doc`, `datasetmethod`, and `eval_results`, among others.

```python
class datalad_next.commands.ValidatedInterface(Bases: Interface)

   Alternative base class for commands with uniform parameter validation

   Note: This interface is a draft. Usage is encouraged, but future changes are to be expected.
```

Commands derived from the traditional `Interface` class have no built-in input parameter validation beyond CLI input validation of individual parameters. Consequently, each command must perform custom parameter validation, which often leads to complex boilerplate code that is largely unrelated to the purpose of a particular command.

This class is part of a framework for uniform parameter validation, regardless of the target API (Python, CLI, GUI). The implementation of a command’s `__call__` method can focus on the core purpose of the command, while validation and error handling can be delegated elsewhere.

A validator for all individual parameters and the joint-set of all parameters can be provided through the `get_parameter_validator()` method.

To transition a command from `Interface` to `ValidatedInterface`, replace the base class declaration and declare a `_validator_` class member. Any `constraints=` declaration for `Parameter` instances should either be removed, or moved to the corresponding entry in `_validator_`.

```python
@classmethod
def get_parameter_validator() -> EnsureCommandParameterization | None

   Returns a validator for the entire parameter set of a command

   If parameter validation shall be performed, this method must return an instance of
   `EnsureCommandParameterization`. All parameters will be passed through this validator, and
   only the its output will be passed to the underlying command’s `__call__` method.

   Consequently, the core implementation of a command only needs to support the output values of the val-
   idators declared by itself.

   Factoring out input validation, normalization, type coercion etc. into a dedicated component also makes it
   accessible for upfront validation and improved error reporting via the different DataLad APIs.

   If a command does not implement parameter validation in this fashion, this method must return `None`.

   The default implementation returns the `_validator_` class member.
```
**datalad_next.config**

Configuration query and manipulation

This module imports the central `ConfigManager` class from DataLad core.

**datalad_next.constraints**

Data validation, coercion, and parameter documentation

This module provides a set of uniform classes to validate and document particular aspects of inputs. In a nutshell, each of these `Constraint` class:

- focuses on a specific aspect, such as data type coercion, or checking particular input properties
- is instantiated with a set of parameters to customize such an instance for a particular task
- performs its task by receiving an input via its `__call__()` method
- provides default auto-documentation that can be customized by wrapping an instance in `WithDescription`

Individual `Constraint` instances can be combined with logical AND (`AllOf`) and OR (`AnyOf`) operations to form arbitrarily complex constructs.

On (validation/coercion) error, instances raise `ConstraintError` via their `raise_for()` method. This approach to error reporting helps to communicate standard (yet customizable) error messages, aids structured error reporting, and is capable of communication the underlying causes of an error in full detail without the need to generate long textual descriptions.

`EnsureCommandParameterization` is a particular variant of a `Constraint` that is capable of validating a complete parameterization of a command (or function), for each parameter individually, and for arbitrary combinations of parameters. It puts a particular emphasis on structured error reporting.

<table>
<thead>
<tr>
<th>base</th>
<th>Base classes for constraints and their logical connectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>Basic constraints for declaring essential data types, values, and ranges</td>
</tr>
<tr>
<td>compound</td>
<td>Constraints that wrap or contain other constraints</td>
</tr>
<tr>
<td>formats</td>
<td>Constraints for particular formats or protocols</td>
</tr>
<tr>
<td>parameter</td>
<td>Constraints for command/function parameters</td>
</tr>
<tr>
<td>git</td>
<td>Constraints for Git-related concepts and parameters</td>
</tr>
<tr>
<td>dataset</td>
<td>Constraints for DataLad datasets</td>
</tr>
<tr>
<td>exceptions</td>
<td>Custom exceptions raised by <code>Constraint</code> implementations</td>
</tr>
</tbody>
</table>

**datalad_next.constraints.base**

Base classes for constraints and their logical connectives

```python
class datalad_next.constraints.base.AllOf(*constraints)
    Bases: _MultiConstraint
    Logical AND for constraints.
    An arbitrary number of constraints can be given. They are evaluated in the order in which they were specified. The return value of each constraint is passed an input into the next. The return value of the last constraint is the global return value. No intermediate exceptions are caught.
    Documentation is aggregated for all constraints.
```

Chapter 2. API
long_description()
    This method is deprecated. Use input_description instead

short_description()
    This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.base.AnyOf(*constraints)
    Bases: _MultiConstraint
    Logical OR for constraints.
    An arbitrary number of constraints can be given. They are evaluated in the order in which they were specified.
    The value returned by the first constraint that does not raise an exception is the global return value.
    Documentation is aggregated for all alternative constraints.

long_description()
    This method is deprecated. Use input_description instead

short_description()
    This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.base.Constraint
    Bases: object
    Base class for value coercion/validation.
    These classes are also meant to be able to generate appropriate documentation on an appropriate parameter value.

for_dataset(dataset: DatasetParameter) → Constraint
    Return a constraint-variant for a specific dataset context
    The default implementation returns the unmodified, identical constraint. However, subclasses can implement different behaviors.

property input_description: str
    Returns full description of valid input for a constraint
    Like input_synopsis this information is user-facing. In contrast, to the synopsis there is length/line limit. Nevertheless, the information should be presented in a compact fashion that avoids needless verbosity. If possible, a single paragraph is a good format. If multiple paragraphs are necessary, they should be separated by a single, empty line.

    Rendering code may indent, or rewrap the text, so no line-by-line formatting will be preserved.

    If possible, the synopsis should be written in a UI/API-agnostic fashion. However, if this is impossible or leads to imprecisions or confusion, it should focus on use within Python code and with Python data types. Tailored documentation can be provided via the WithDescription wrapper.

property input_synopsis: str
    Returns brief, single line summary of valid input for a constraint
    This information is user-facing, and to be used in any place where space is limited (tooltips, usage summaries, etc).

    If possible, the synopsis should be written in a UI/API-agnostic fashion. However, if this is impossible or leads to imprecisions or confusion, it should focus on use within Python code and with Python data types. Tailored documentation can be provided via the WithDescription wrapper.
long_description()
This method is deprecated. Use input_description instead

raise_for(value, msg, **ctx) → None
Convenience method for raising a ConstraintError
The parameters are identical to those of ConstraintError. This method merely passes the Constraint instance as self to the constructor.

short_description()
This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.base.DatasetParameter(original, ds)
Bases: object
Utility class to report an original and resolve dataset parameter value
This is used by EnsureDataset to be able to report the original argument semantics of a dataset parameter to a receiving command. It is consumed by any Constraint.for_dataset().
The original argument is provided via the original property. A corresponding Dataset instance is provided via the ds property.

datalad_next.constraints.basic

Basic constraints for declaring essential data types, values, and ranges

class datalad_next.constraints.basic.EnsureBool
Bases: Constraint
Ensure that an input is a bool.
A couple of literal labels are supported, such as: False: '0', 'no', 'off', 'disable', 'false' True: '1', 'yes', 'on', 'enable', 'true'

long_description()
This method is deprecated. Use input_description instead

short_description()
This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.basic.EnsureCallable
Bases: Constraint
Ensure an input is a callable object

long_description()
This method is deprecated. Use input_description instead

short_description()
This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.basic.EnsureChoice(*values)
Bases: Constraint
Ensure an input is element of a set of possible values

long_description()
This method is deprecated. Use input_description instead
This method is deprecated. Use `input_synopsis` instead

```python
>>> c = EnsureDType(float)
>>> type(c(8))
float
>>> import numpy as np
>>> c = EnsureDType(np.float64)
>>> type(c(8))
numpy.float64
```

This method is deprecated. Use `input_synopsis` instead

**class** `datalad_next.constraints.basic.EnsureDType(dtype)`

Bases: `Constraint`

Ensure that an input (or several inputs) are of a particular data type.

**long_description()**

This method is deprecated. Use `input_description` instead

**short_description()**

This method is deprecated. Use `input_synopsis` instead

**class** `datalad_next.constraints.basic.EnsureFloat`

Bases: `EnsureDType`

Ensure that an input (or several inputs) are of a data type `float`.

**class** `datalad_next.constraints.basic.EnsureInt`

Bases: `EnsureDType`

Ensure that an input (or several inputs) are of a data type `int`.

**class** `datalad_next.constraints.basic.EnsureKeyChoice(key, values)`

Bases: `EnsureChoice`

Ensure value under a key in an input is in a set of possible values

**long_description()**

This method is deprecated. Use `input_description` instead

**short_description()**

This method is deprecated. Use `input_synopsis` instead

**class** `datalad_next.constraints.basic.EnsureNone`

Bases: `EnsureValue`

Ensure an input is of value `None`

**class** `datalad_next.constraints.basic.EnsurePath(*, path_type: type = <class 'pathlib.Path'>, is_format: str | None = None, leexists: bool | None = None, is_mode: callable | None = None, ref: Path | None = None, ref_is: str = 'parent-or-same-as', dsarg: DatasetParameter | None = None)`

Bases: `Constraint`

Ensures input is convertible to a (platform) path and returns a `Path`

Optionally, the path can be tested for existence and whether it is absolute or relative.
**for_dataset**(*dataset: DatasetParameter*) → *Constraint*

Return an similarly parametrized variant that resolves paths against a given dataset (argument)

**short_description**()

This method is deprecated. Use *input_synopsis* instead

**class** datalad_next.constraints.basic.*EnsureRange*(min=None, max=None)

Bases: *Constraint*

Ensure an input is within a particular range

No type checks are performed.

**long_description**()

This method is deprecated. Use *input_description* instead

**short_description**()

This method is deprecated. Use *input_synopsis* instead

**class** datalad_next.constraints.basic.*EnsureStr*(min_len: int = 0, match: str | None = None)

Bases: *Constraint*

Ensure an input is a string of some min. length and matching a pattern

Pattern matching is optional and minimum length is zero (empty string is OK).

No type conversion is performed.

**long_description**()

This method is deprecated. Use *input_description* instead

**short_description**()

This method is deprecated. Use *input_synopsis* instead

**class** datalad_next.constraints.basic.*EnsureStrPrefix*(prefix)

Bases: *EnsureStr*

Ensure an input is a string that starts with a given prefix.

**long_description**()

This method is deprecated. Use *input_description* instead

**short_description**()

This method is deprecated. Use *input_synopsis* instead

**class** datalad_next.constraints.basic.*EnsureValue*(value)

Bases: *Constraint*

Ensure an input is a particular value

**long_description**()

This method is deprecated. Use *input_description* instead

**short_description**()

This method is deprecated. Use *input_synopsis* instead

**class** datalad_next.constraints.basic.*NoConstraint*

Bases: *Constraint*

A constraint that represents no constraints

**short_description**()

This method is deprecated. Use *input_synopsis* instead
**datalad_next.constraints.compound**

Constraints that wrap or contain other constraints

```python
class datalad_next.constraints.compound.ConstraintWithPassthrough(constraint: Constraint, passthrough: Any)
```

**Bases:** `Constraint`

Regular constraint, but with a "pass-through" value that is not processed

This is different from a `Constraint() | EnsureValue(...)` construct, because the pass-through value is not communicated. This can be useful when a particular value must be supported for technical reasons, but need not, or must not be included in (error) messages.

The pass-through is returned as-is, and is not processed except for an identity check (==).

For almost all reporting (__str__, descriptions, ...) the wrapped value constraint is used, making this class virtually invisible. Only __repr__ reflects the wrapping.

**property constraint:** `Constraint`

Returns the wrapped constraint instance

**for_dataset** *(dataset: DatasetParameter) → Constraint*

Wrap the wrapped constraint again after tailoring it for the dataset

The pass-through value is re-used.

**long_description() → str**

This method is deprecated. Use `input_description` instead

**property passthrough:** `Any`

Returns the set pass-through value

**short_description() → str**

This method is deprecated. Use `input_synopsis` instead

```python
class datalad_next.constraints.compound.EnsureGeneratorFromFileLike(item_constraint: Callable, exec_mode: str = 'raise')
```

**Bases:** `Constraint`

Ensure a constraint for each item read from a file-like.

A given value can either be a file-like (the outcome of `open()`), or `StringIO`, or - as an alias of STDIN, or a path to an existing file to be read from.

**short_description()**

This method is deprecated. Use `input_synopsis` instead

```python
class datalad_next.constraints.compound.EnsureIterableOf(iter_type: type, item_constraint: Callable, min_len: int | None = None, max_len: int | None = None)
```

**Bases:** `Constraint`

Ensure that an input is a list of a particular data type

**property item_constraint**

| short_description() |

This method is deprecated. Use `input_synopsis` instead

2.3. Python tooling
class datalad_next.constraints.compound.EnsureListOf(
    item_constraint: Callable, min_len: int | None = None, max_len: int | None = None
)

Bases: EnsureIterableOf

short_description()

This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.compound.EnsureMapping(
    key: Constraint, value: Constraint, delimiter: str = ':', allow_length2_sequence: bool = True
)

Bases: Constraint

Ensure a mapping of a key to a value of a specific nature

for_dataset(dataset: DatasetParameter) → Constraint

Return a constraint-variant for a specific dataset context

The default implementation returns the unmodified, identical constraint. However, subclasses can implement different behaviors.

short_description()

This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.compound.EnsureTupleOf(
    item_constraint: Callable, min_len: int | None = None, max_len: int | None = None
)

Bases: EnsureIterableOf

short_description()

This method is deprecated. Use input_synopsis instead

class datalad_next.constraints.compound.WithDescription(
    constraint: Constraint, *, input_synopsis: str | None = None, input_description: str | None = None, error_message: str | None = None, input_synopsis_for_ds: str | None = None, input_description_for_ds: str | None = None, error_message_for_ds: str | None = None
)

Bases: Constraint

Constraint that wraps another constraint and replaces its description

Whenever a constraint’s self-description does not fit an application context, it can be wrapped with this class. The given synopsis and description of valid inputs replaces those of the wrapped constraint.

property constraint: Constraint

Returns the wrapped constraint instance

for_dataset(dataset: DatasetParameter) → Constraint

Wrap the wrapped constraint again after tailoring it for the dataset

property input_description

Returns full description of valid input for a constraint

Like input_synopsis this information is user-facing. In contrast, to the synopsis there is length/line limit. Nevertheless, the information should be presented in a compact fashion that avoids needless verbosity. If possible, a single paragraph is a good format. If multiple paragraphs are necessary, they should be separated by a single, empty line.

Rendering code may indent, or rewrap the text, so no line-by-line formatting will be preserved.
If possible, the synopsis should be written in a UI/API-agnostic fashion. However, if this is impossible or leads to imprecisions or confusion, it should focus on use within Python code and with Python data types. Tailored documentation can be provided via the WithDescription wrapper.

**property input_synopsis**
Returns brief, single line summary of valid input for a constraint

This information is user-facing, and to be used in any place where space is limited (tooltips, usage summaries, etc).

If possible, the synopsis should be written in a UI/API-agnostic fashion. However, if this is impossible or leads to imprecisions or confusion, it should focus on use within Python code and with Python data types. Tailored documentation can be provided via the WithDescription wrapper.

**long_description() → str**
This method is deprecated. Use input_description instead

**short_description() → str**
This method is deprecated. Use input_synopsis instead

### datalad_next.constraints.formats

Constraints for particular formats or protocols

**class** datalad_next.constraints.formats.EnsureJSON

Bases: Constraint

Ensures that string is JSON formatted and can be deserialized.

**short_description()**
This method is deprecated. Use input_synopsis instead

**class** datalad_next.constraints.formats.EnsureParsedURL(required: list | None = None, forbidden: list | None = None, match: str | None = None)

Bases: EnsureURL

Like EnsureURL, but returns a parsed URL

**class** datalad_next.constraints.formats.EnsureURL(required: list | None = None, forbidden: list | None = None, match: str | None = None)

Bases: Constraint

Ensures that a string is a valid URL with a select set of components and/or:

- does not contain certain components
- matches a particular regular expression

Given that a large variety of strings are also a valid URL, a typical use of this constraint would involve using a `required=['scheme']` setting.

All URL attribute names supported by `urllib.parse.urlparse()` are also supported here: scheme, netloc, path, params, query, fragment, username, password, hostname, port.

**See also:**

https://docs.python.org/3/library/urllib.parse.html#urllib.parse.urlparse
This method is deprecated. Use `input_synopsis` instead

### `datalad_next.constraints.parameter` Constraints for command/function parameters

```python
class datalad_next.constraints.parameter.EnsureCommandParameterization(param_constraints:
    Dict[str, Constraint], *,
    validate_defaults:
    Container[str] | None = None,
    joint_constraints:
    Dict[ParameterConstraintContext, Callable] | None = None,
    tailor_for_dataset:
    Dict[str, str] | None = None)
```

**Bases:** `Constraint`

Base class for `ValidatedInterface` parameter validators

This class can be used as-is, by declaring individual constraints in the constructor, or it can be subclassed to consolidate all custom validation-related code for a command in a single place.

Commonly this constraint is used by declaring particular value constraints for individual parameters as a mapping. Declaring that the `path` parameter should receive something that is or can be coerced to a valid `Path` object looks like this:

```python
EnsureCommandParameterization({'path': EnsurePath()})
```

This class differs from a standard `Constraint` implementation, because its `__call__()` method supports additional arguments that are used by the internal `Interface` handling code to control how parameters are validated.

During validation, when no validator for a particular parameter is declared, any input value is passed on as-is, and otherwise an input is passed through the validator.

There is one exception to this rule: When a parameter value is identical to its default value (as declared in the command signature, and communicated via the `at_default` argument of `__call__()`), this default value is also passed as-is, unless the respective parameter name is included in the `validate_defaults` constructor argument.

An important consequence of this behavior is that validators need not cover a default value. For example, a parameter constraint for `path=None`, where `None` is a special value used to indicate an optional and unset value, but actually only paths are acceptable input values, can simply use `EnsurePath()` and it is not necessary to do something like `EnsurePath() | EnsureNone()`.

However, `EnsureCommandParameterization` can also be specifically instructed to perform validation of defaults for individual parameters, as described above. A common use case is the auto-discovery of datasets, where often `None` is the default value of a `dataset` parameter (to make it optional), and an `EnsureDataset` constraint is used. This constraint can perform the auto-discovery (with the `None` value indicating that), but validation of defaults must be turned on for the `dataset` parameter in order to do that.

A second difference to a common `Constraint` implementation is the ability to perform an "exhaustive validation" on request (via `__call__`(on_error=...)). In this case, validation is not stopped at the first discovered violation, but all violations are collected and communicated by raising a `CommandParametrizationError` exception, which can be inspected by a caller for details on number and nature of all discovered violations.
Exhaustive validation and joint reporting are only supported for individual constraint implementations that raise `ConstraintError` exceptions. For legacy constraints, any raised exception of another type are not caught and reraised immediately.

```python
joint_validation(params: Dict, on_error: str) → Dict
```

Higher-order validation considering multiple parameters at a time.

This method is called with all, individually validated, command parameters in keyword-argument form in the `params` dict argument.

Arbitrary additional validation steps can be performed on the full set of parameters that may involve raising exceptions on validation errors, but also value transformation or replacements of individual parameters based on the setting of others.

The parameter values returned by the method are passed on to the respective command implementation.

The default implementation iterates over the `joint_validators` specification given to the constructor, in order to perform any number of validations. This is a mapping of a `ParameterConstraintContext` instance to a callable implementing a validation for a particular parameter set.

Example:

```python
_joint_validators_ = {
    ParameterConstraintContext(('p1', 'p2'), 'sum'): MyValidator._check_sum,
}

def _checksum(self, p1, p2):
    if (p1 + p2) < 3:
        self.raise_for(
            dict(p1=p1, p2=p2),
            'parameter sum is too large',
        )
```

The callable will be passed the arguments named in the `ParameterConstraintContext` as keyword arguments, using the same names as originally given to `EnsureCommandParameterization`.

Any raised `ConstraintError` is caught and reported together with the respective `ParameterConstraintContext`. The violating value reported in such a `ConstraintError` must be a mapping of parameter name to value, comprising the full parameter set (i.e., keys matching the `ParameterConstraintContext`). The use of `self.raise_for()` is encouraged.

If the callable anyhow modifies the passed arguments, it must return them as a kwargs-like mapping. If nothing is modified, it is OK to return `None`.

**Returns**

- `dict` -- The returned dict must have a value for each item passed in via `params`.
- `on_error` (["raise-early", "raise-at-end"]) -- Flag how to handle constraint violation. By default, validation is stopped at the first error and an exception is raised. When an exhaustive validation is performed, an eventual exception contains information on all constraint violations.

**Raises**

- `ConstraintErrors` -- With `on_error='raise-at-end'` an implementation can choose to collect more than one higher-order violation and raise them as a `ConstraintErrors` exception.

```python
class datalad_next.constraints.parameter.NoValue
    Bases: object
```

**2.3. Python tooling**
Type to annotate the absence of a value

For example in a list of parameter defaults. In general None cannot be used, as it may be an actual value, hence we use a local, private type.

datalad_next.constraints.git

Constraints for Git-related concepts and parameters

class datalad_next.constraints.git.EnsureGitRefName(allow_onelevel: bool = True, normalize: bool = True, refspec_pattern: bool = False)

Bases: Constraint

Ensures that a reference name is well formed

Validation is performed by calling git check-ref-format.

short_description()

This method is deprecated. Use input_synopsis instead

datalad_next.constraints.dataset

Constraints for DataLad datasets

class datalad_next.constraints.dataset.EnsureDataset(installed: bool | None = None, purpose: str | None = None, require_id: bool | None = None)

Bases: Constraint

Ensure an absent/present Dataset from any path or Dataset instance

Regardless of the nature of the input (Dataset instance or local path) a resulting instance (if it can be created) is optionally tested for absence or presence on the local file system.

Due to the particular nature of the Dataset class (the same instance is used for a unique path), this constraint returns a DatasetParameter rather than a Dataset directly. Consuming commands can discover the original parameter value via its original property, and access a Dataset instance via its ds property.

In addition to any value representing an explicit path, this constraint also recognizes the special value None. This instructs the implementation to find a dataset that contains the process working directory (PWD). Such a dataset need not have its root at PWD, but could be located in any parent directory too. If no such dataset can be found, PWD is used directly. Tests for installed are performed in the same way as with an explicit dataset location argument. If None is given and installed=True, but no dataset is found, an exception is raised (this is the behavior of the required_dataset() function in the DataLad core package). With installed=False no exception is raised and a dataset instances matching PWD is returned.

short_description() → str

This method is deprecated. Use input_synopsis instead
**datalad_next.constraints.exceptions**

Custom exceptions raised by `Constraint` implementations

**exception** `datalad_next.constraints.exceptions.CommandParametrizationError()`

```python
exceptions:
    Dict[str, ConstraintError] | Dict[ParameterConstraintContext, ConstraintError])
```

Bases: `ParametrizationErrors`

Exception type raised on violating any command parameter constraints

**See also:**
`EnsureCommandParameterization`

**exception** `datalad_next.constraints.exceptions.ConstraintError()`

```python
constraint, value: Any, msg: str, ctx: Dict[str, Any] | None = None)
```

Bases: `ValueError`

Exception type raised by constraints when their conditions are violated

A primary purpose of this class is to provide uniform means for communicating information on violated con-
straints.

**property caused_by:** `Tuple[Exception] | None`

Returns a tuple of any underlying exceptions that caused a violation

**property constraint**

Get the instance of the constraint that was violated

**property context:** `mappingproxy`

Get a constraint violation's context

This is a mapping of key/value-pairs matching the `ctx` constructor argument.

**property msg**

Obtain an (interpolated) message on the constraint violation

The error message template can be interpolated with any information available in the error context dict
(`ctx`). In addition to the information provided by the `Constraint` that raised the error, the following
additional placeholders are provided:

- **__value__**: the value reported to have caused the error
- **__itemized-causes__**: an indented bullet list str with on item for each error in the `caused_by`
  report of the error.

Message template can use any feature of the Python format mini language. For example `{__value__!r}`
to get a `repr()`-style representation of the offending value.

**property value**

Get the value that violated the constraint

**exception** `datalad_next.constraints.exceptions.ConstraintErrors()`

```python
exceptions: Dict[Any, ConstraintError])
```

Bases: `ConstraintError`

Exception representing context-specific `ConstraintError` instances

---

**2.3. Python tooling** 45
This class enables the association of a context in which any particular constraint was violated. This is done by passing a mapping, of a context identifier (e.g., a label) to the particular ConstraintError that occurred in this context, to the constructor.

This is a generic implementation with no requirements regarding the nature of the context identifiers (expect for being hashable). See CommandParametrizationError for a specialization.

```python
property errors: mappingproxy[Any, datalad_next.constraints.exceptions.ConstraintError]
```

```python
class datalad_next.constraints.exceptions.ParameterConstraintContext(parameters: Tuple[str],
description: str | None = None)
```

Bases: object

Representation of a parameter constraint context

This type is used for the keys in the error map of. ParametrizationErrors. Its purpose is to clearly identify which parameter combination (and its nature) led to a ConstraintError.

An error context comprises to components: 1) the names of the parameters that were considered, and 2) a description of how the parameters were linked or combined. In the simple case of an error occurring in the context of a single parameter, the second component is superfluous. Otherwise, it can be thought of as an operation label, describing what aspect of the set of parameters is being relevant in a particular context.

Example:

A command has two parameters \( p_1 \) and \( p_2 \). The may also have respective individual constraints, but importantly they 1) must not have identical values, and 2) their sum must be larger than 3. If the command is called with \( \text{cmd}(p_1=1, p_2=1) \), both conditions are violated. The reporting may be implemented using the following ParameterConstraintContext and ConstraintError instances:

```python
ParameterConstraintContext(('p1', 'p2'), 'inequality):
    ConstraintError(EnsureValue(True), False, <EnsureValue error>)
```

```python
ParameterConstraintContext(('p1', 'p2'), 'sum):
    ConstraintError(EnsureRange(min=3), False, <EnsureRange error>)
```

where the ConstraintError instances are generated by standard Constraint implementation. For the second error, this could look like:

```python
EnsureRange(min=3)(params['p1'] + params['p2'])
```

```python
description: str | None = None
```

```python
get_label_with_parameter_values(values: dict) → str
```

Like .label but each parameter will also state a value

```python
property label: str
```

A concise summary of the context

This label will be a compact as possible.

```python
parameters: Tuple[str]
```

```python
class datalad_next.constraints.exceptions.ParameterContextErrors(errors:
    Dict[ParameterConstraintContext, ConstraintError])
```

46 Chapter 2. API
Bases: Mapping
Read-only convenience that wraps a ConstraintErrors error mapping

property context_labels
items() → a set-like object providing a view on D’s items

property messages

exception datalad_next.constraints.exceptions.ParametrizationErrors(exceptions: Dict[str, ConstraintError] | Dict[ParameterConstraintContext, ConstraintError])

Bases: ConstraintErrors
Exception type raised on violating parameter constraints

This is a ConstraintErrors variant that uses parameter names (i.e., str labels) as context identifiers. In addition to individual parameter names an additional __all__ identifier is recognized. It can be used to record a ConstraintError arising from high-order constraints, such as the violation of "mutually exclusive" requirements across more than one parameter.

property errors: ParameterContextErrors

datalad_next.credman

Credential management

manager Credential management and query

datalad_next.credman.manager

Credential management and query

class datalad_next.credman.manager.CredentialManager(cfg: ConfigManager | None = None)
Bases: object
Facility to get, set, remove and query credentials.

A credential in this context is a set of properties (key-value pairs) associated with exactly one secret.

At present, the only backend for secret storage is the Python keyring package, as interfaced via a custom DataLad wrapper. Store for credential properties is implemented using DataLad’s (i.e. Git’s) configuration system. All properties are stored in the global (i.e., user) scope under configuration items following the pattern:

datalad.credential.<name>.<property>

where <name> is a credential name/identifier, and <property> is an arbitrarily named credential property, whose name must follow the git-config syntax for variable names (case-insensitive, only alphanumeric characters and -, and must start with an alphabetic character).

Create a CredentialManager instance is fast, virtually no initialization needs to be performed. All internal properties are lazily evaluated. This facilitates usage in code where it is difficult to incorporate a long-lived central instance.

2.3. Python tooling
API

With one exception, all parameter names of methods in the core API outside **kwargs must have a _ prefix that distinguishes credential properties from method parameters. The one exception is the name parameter, which is used as a primary identifier (albeit being optional for some operations).

The obtain() method is provided as an additional convenience, and implements a standard workflow for obtaining a credential in a wide variety of scenarios (credential name, credential properties, secret either respectively already known or yet unknown).

**get**(name=None, *, _prompt=None, _type_hint=None, **kwargs)

Get properties and secret of a credential.

This is a read-only method that never modifies information stored on a credential in any backend.

Credential property lookup is supported via a number of approaches. When providing name, all existing corresponding configuration items are found and reported, and an existing secret is retrieved from name-based secret backends (presently keyring). When providing a type property or a _type_hint the lookup of additional properties in the keyring-backend is enabled, using predefined property name lists for a number of known credential types.

For all given property keys that have no value assigned after the initial lookup, manual/interactive entry is attempted, whenever a custom _prompt was provided. This include requesting a secret. If manually entered information is contained in the return credential record, the record contains an additional _edited property with a value of True.

If no secret is known after lookup and a potential manual data entry, a plain None is returned instead of a full credential record.

**Parameters**

- **name** (str, optional) -- Name of the credential to be retrieved
- **_prompt** (str or None) -- Instructions for credential entry to be displayed when missing properties are encountered. If None, manual entry is disabled.
- **_type_hint** (str or None) -- In case no type property is included in kwargs, this parameter is used to determine a credential type, to possibly enable further lookup/entry of additional properties for a known credential type
- ****kwargs **-- Credential property name/value pairs to overwrite/amend potentially existing properties. For any property with a value of None, manual data entry will be performed, unless a value could be retrieved on lookup, or prompting was not enabled.

**Returns**

Return None, if no secret for the credential was found or entered. Otherwise returns the complete credential record, comprising all properties and the secret. An additional _edited key with a value of True is added whenever the returned record contains manually entered information.

**Return type**

dict or None

**Raises**

ValueError -- When the method is called without any information that could be used to identify a credential

**obtain**(name: str | None = None, *, prompt: str | None = None, type_hint: str | None = None, query_props: Dict | None = None, expected_props: List | Tuple | None = None)

Obtain a credential by query or prompt (if needed)
This convenience method implements a standard workflow to obtain a credential. It supports credential selection by credential name/identifier, and falls back onto querying for a credential matching a set of specified properties (as key-value mappings). If no suitable credential is known, a user is prompted to enter one interactively (if possible in the current session).

If a credential was entered manually, any given type_hint will be included as a type property of the returned credential, and the returned credential has an _edited=True property. Likewise, any realm property included in the query_props is included in the returned credential in this case.

If desired, a credential workflow can be completed, after a credential was found to be valid/working, by storing or updating it in the credential store:

```python
cm = CredentialManager()
cname, cprops = cm.obtain(...)  # verify credential is working
...
# set/update
cm.set(cname, _lastused=True, **cprops)
```

In the code sketch above, if `cname` is `None` (as it will be for a newly entered credential, `set()` will prompt for a name to store the credential under, and will offer a user the choice to skip storing a credential. For any previously known credential, the last-used property will be updated to enable preferred selection in future credential discovery attempts via `obtain()`.

### Examples

Minimal call to get a credential entered (manually):

```python
credman.obtain(type_hint='token', prompt='Credential please!')
```

Without a prompt text no interaction is attempted, and without a type hint it is unknown what (and how much) to enter.

Minimal call to retrieve a credential by its identifier:

```python
credman.obtain('my-github-token')
```

Minimal call to retrieve the last-used credential for a particular authentication "realm". In this case "realm" is a property that was previously set to match a particular service/location, and is now used to match credentials against:

```python
credman.obtain(query_props={'realm': 'mysecretlair'})
```

### Parameters

- `name (str, optional)` -- Name of the credential to be retrieved
- `prompt (str, optional)` -- Passed to `CredentialManager.get()` if a credential name was provided, or no suitable credential could be found by querying.
- `type_hint (str, optional)` -- In case no type property is included in query_props, this parameter is passed to `CredentialManager.get()`.
- `query_props (dict, optional)` -- Credential property to be used for querying for a suitable credential. When multiple credentials match a query, the last-used credential is selected.
• `expected_props(list or tuple, optional)` -- When specified, a credential will be inspected to contain properties matching all listed property names, or a `ValueError` will be raised.

**Returns**
Credential name (possibly different from the input, when a credential was discovered based on properties), and credential properties.

**Return type**
(str, dict)

**Raises**
`ValueError` -- Raised when no matching credential could be found and none was entered. Also raised, when a credential selected from a query result or a manually entered one is missing any of the properties with a name given in `expected_props`.

`query_(**kwargs)`
Query for all (matching) credentials.

This method cannot find credentials for which only a secret was deposited in the keyring. This method does support lookup of credentials defined in DataLad's "provider" configurations.

**Parameters**

• `**kwargs` -- If not given, any found credential is yielded. Otherwise, any credential must match all property name/value pairs

**Yields**
`tuple(str, dict)` -- The first element in the tuple is the credential name, the second element is the credential record as returned by `get()` for any matching credential.
remove(name, *, type_hint=None)

Remove a credential, including all properties and secret

Presently, all supported backends require the specification of a credential name for lookup. This may change in the future, when support for alternative backends is added, at which point the name parameter would become optional, and additional parameters would be added.

Returns
True if a credential was removed, and False if not (because no respective credential was found).

Return type
bool

Raises
RuntimeError -- This exception is raised whenever a property cannot be removed successfully. Likely cause is that it is defined in a configuration scope or backend for which write-access is not supported.

secret_names = { 'user_password': 'password' }

set(name, *, _lastused=False, _suggested_name=None, _context=None, **kwargs)

Set credential properties and secret

Presently, all supported backends require the specification of a credential name for storage. This may change in the future, when support for alternative backends is added, at which point the name parameter would become optional.

All properties provided as kwargs with keys not starting with _ and with values that are not None will be stored. If kwargs do not contain a secret specification, manual entry will be attempted. The associated prompt with be either the name of the secret field of a known credential (as identified via a type property), or the label 'secret'.

All properties with an associated value of None will be removed (unset).

Parameters

- name (str or None) -- Credential name. If None, the name will be prompted for and setting the credential is skipped if no name is provided.
- _lastused (bool, optional) -- If set, automatically add an additional credential property 'last-used' with the current timestamp in ISO 8601 format.
- _suggested_name (str, optional) -- If name is None, this name (if given) is presented as a default suggestion that can be accepted without having to enter it manually. If this name suggestion conflicts with an existing credential, it is ignored and not presented as a suggestion.
- _context (str, optional) -- If given, will be included in the prompt for a missing credential name to provide context for a user. It should be written to fit into a parenthetical statement after "Enter a name to save the credential (…)", e.g. "for download from <URL>".
- **kwargs -- Any number of credential property key/value pairs to set (update), or remove. With one exception, values of None indicate removal of a property from a credential. However, secret=None does not lead to the removal of a credential's secret, because it would result in an incomplete credential. Instead, it will cause a credential's effective secret property to be written to the secret store. The effective secret might come from other sources, such as particular configuration scopes or environment variables (i.e., matching the datalad.credential.<name>.secret configuration item. Properties whose names start with an underscore are automatically removed prior storage.

2.3. Python tooling
Returns
key/values of all modified credential properties with respect to their previously recorded values. None is returned in case a user did not enter a missing credential name. If a user entered a credential name, it is included in the returned dictionary under the 'name' key.

Return type
dict or None

Raises
• RuntimeError -- This exception is raised whenever a property cannot be removed successfully. Likely cause is that it is defined in a configuration scope or backend for which write-access is not supported.
• ValueError -- When property names in kwargs are not syntax-compliant.

valid_property_names_regex = re.compile('[a-z0-9]+[a-z0-9-]*$')

datalad_next.datasets

Representations of DataLad datasets built on git/git-annex repositories

Two sets of repository abstractions are available LeanGitRepo and LeanAnnexRepo vs. LegacyGitRepo and LegacyAnnexRepo.

The latter are the classic classes providing a, now legacy, low-level API to repository operations. This functionality stems from the earliest days of DataLad and implements paradigms and behaviors that are no longer common to the rest of the DataLad API. LegacyGitRepo and LegacyAnnexRepo should no longer be used in new developments.

LeanGitRepo and LeanAnnexRepo on the other hand provide a more modern, substantially restricted API and represent the present standard API for low-level repository operations. They are geared towards interacting with Git and git-annex more directly, and are more suitable for generator-like implementations, promoting low response latencies, and a leaner processing footprint.

class datalad_next.datasets.LeanAnnexRepo(*args, **kwargs)
   Bases: AnnexRepo
   git-annex repository representation with a minimized API
   This is a companion of LeanGitRepo. In the same spirit, it restricts its API to a limited set of method that primarily extend LeanGitRepo with a set of call_annex*() methods.

datalad_next.exceptions

All custom exceptions used in datalad-next

datalad_next.iter_collections

Iterators for particular types of collections

Most importantly this includes different collections (or containers) for files, such as a file system directory, or an archive (also see the ls_file_collection command). However, this module is not per-se limited to file collections.

Most, if not all, implementation come in the form of a function that takes a collection identifier or a collection location (e.g., a file system path), and possibly some additional options. When called, an iterator is returned that produces collection items in the form of data class instances of a given type. The particular type can be different across different collections.
**datalad_next.iter_collections.directory**

Report on the content of directories

The main functionality is provided by the `iter_dir()` function.

```python
class datalad_next.iter_collections.directory.DirectoryItem:
    type: FileSystemItemType
    size: int
    name: PurePath
    mtime: float
    mode: int
    uid: int
    gid: int
    link_target: PurePath
    fp: IO

Bases: FileSystemItem
```

`datalad_next.iter_collections.directory.iter_dir(path: Path, *, fp: bool = False) -> Generator[DirectoryItem, None, None]`

Uses `Path.iterdir()` to iterate over a directory and reports content

The iterator produces a `DirectoryItem` instance with standard information on file system elements, such as size, or mtime.

In addition to a plain `Path.iterdir()` the report includes a path-type label (distinguished are `file`, `directory`, `symlink`).

**Parameters**

- `path (Path)` -- Path of the directory to report content for (iterate over).
- `fp (bool, optional)` -- If `True`, each file-type item includes a file-like object to access the file's content. This file handle will be closed automatically when the next item is yielded.

**Yields**

`DirectoryItem`

**datalad_next.iter_collections.gitworktree**

Report on the content of a Git repository worktree

The main functionality is provided by the `iter_gitworktree()` function.

```python
class datalad_next.iter_collections.gitworktree.GitTreeItemType:
    value

Bases: Enum
```

Enumeration of item types of Git trees
directory = 'directory'
executablefile = 'executablefile'
file = 'file'
submodule = 'submodule'
symlink = 'symlink'

class datalad_next.iter_collections.gitworktree.GitWorktreeFileSystemItem(
type: 'FileSystemItemType',
nname: 'PurePath',
ssize: 'int', mtime: 'float | None' = None,
mode: 'int | None' = None, uid: 'int | None' = None,
gid: 'int | None' = None, link_target: 'PurePath | None' = None,
fp: 'IO | None' = None,
gitsha: 'str | None' = None,
gittype: 'GitTreeItemType | None' = None)

Bases: FileSystemItem

gitsha: str | None = None
gittype: GitTreeItemType | None = None

class datalad_next.iter_collections.gitworktree.GitWorktreeItem(
nname: 'PurePath', gitsha: 'str | None' = None,
gittype: 'GitTreeItemType | None' = None)

Bases: PathBasedItem

gitsha: str | None = None
gittype: GitTreeItemType | None = None

name: PurePath

datalad_next.iter_collections.gitworktree.iter_gitworktree(
path: Path, *, untracked: str | None = 'all', link_target: bool = False, fp: bool = False) →
Generator[GitWorktreeItem | GitWorktreeFileSystemItem, None, None]

Uses git ls-files to report on a work tree of a Git repository

This iterator can be used to report on all tracked, and untracked content of a Git repository's work tree. This includes files that have been removed from the work tree (deleted), unless their removal has already been staged.
For any tracked content, yielded items include type information and `gitsha` as last known to Git. This means that such reports reflect the last committed or staged content, not the state of a potential unstaged modification in the work tree.

When no reporting of link targets or file-objects are requested, items of type `GitWorktreeItem` are yielded, otherwise `GitWorktreeFileSystemItem` instances. In both cases, `gitsha` and `gittype` properties are provided. Either of them being `None` indicates untracked work tree content.

**Note:** The `gitsha` is not equivalent to a SHA1 hash of a file's content, but is the SHA-type blob identifier as reported and used by Git.

**Parameters**

- **`path` (Path)** -- Path of a directory in a Git repository to report on. This directory need not be the root directory of the repository, but must be part of the repository's work tree.

- **`untracked` ({`all`, `whole-dir`, `no-empty`) or `None`, optional}** -- If not `None`, also reports on untracked work tree content. All reports on any untracked file; `whole-dir` yields a single report for a directory that is entirely untracked, and not individual untracked files in it; `no-empty-dir` skips any reports on untracked empty directories. Any untracked content is yielded as a `PurePosixPath`.

- **`link_target` (bool, optional)** -- If `True`, each file-type item includes a file-like object to access the file's content. This file handle will be closed automatically when the next item is yielded.

- **`fp` (bool, optional)** -- If `True`, each file-type item includes a file-like object to access the file's content. This file handle will be closed automatically when the next item is yielded.

**Yields**

`GitWorktreeItem` or `GitWorktreeFileSystemItem`

datalad_next.iter_collections.tarfile

Report on the content of TAR archives

The main functionality is provided by the `iter_tar()` function.

**class datalad_next.iter_collections.tarfile.TarfileItem**

```python
type: 'FileSystemItemType',
name: 'PurePosixPath',
size: 'int',
mode: 'int |
uid: 'int |
gid: 'int |
ltrim: 'float |
None' = None,
link_target: 'PurePosixPath | None' = None,
uid: 'int |
None' = None,
gid: 'int |
None' = None,
ltrim: 'float |
None' = None
```

Bases: `FileSystemItem`

- **link_target**: `PurePosixPath | None = None`  
  Just as for `name`, a link target is also reported in POSIX format.

- **name**: `PurePosixPath`  
  TAR uses POSIX paths as item identifiers. Not all POSIX paths can be represented on all (non-POSIX) file systems, therefore the item name is represented in POSIX form, instead of a platform-dependent `PurePath`.

datalad_next.iter_collections.tarfile.iter_tar

```python
(path: Path, *, fp: bool = False) →
Generator[TarfileItem, None, None]
```

2.3. Python tooling 55
Uses the standard library tarfile module to report on TAR archives.

A TAR archive can represent more or less the full bandwidth of file system properties, therefore reporting on archive members is implemented similar to \texttt{iter\_dir()}. The iterator produces an \texttt{TarfileItem} instance with standard information on file system elements, such as size, or \texttt{mtime}.

**Parameters**

- \texttt{path (Path)} -- Path of the TAR archive to report content for (iterate over).
- \texttt{fp (bool, optional)} -- If True, each file-type item includes a file-like object to access the file's content. This file handle will be closed automatically when the next item is yielded or the function returns.

**Yields**

\texttt{TarfileItem}

datalad\_next.iter\_collections.zipfile

Report on the content of ZIP file

The main functionality is provided by the \texttt{iter\_zip()} function.

**class datalad\_next.iter\_collections.zipfile.ZipfileItem**

\begin{verbatim}
(type: 'FileSystemItemType', name: 'PurePosixPath', size: 'int', mtime: 'float | None = None', uid: 'int | None' = None, gid: 'int | None' = None, link_target: 'PurePath | None' = None, fp: 'IO | None' = None)
\end{verbatim}

**Bases:** \texttt{FileSystemItem}

**name:** PurePosixPath

ZIP uses POSIX paths as item identifiers from version 6.3.3 onwards. Not all POSIX paths are legal paths on non-POSIX file systems or platforms. Therefore we cannot use a platform-dependent PurePath-instance to address ZIP-file items, anq we use PurePosixPath-instances instead.

datalad\_next.iter\_collections.zipfile.iter\_zip(path: Path, *, fp: bool = False) →

Generator[ZipfileItem, None, None]

Uses the standard library zipfile module to report on ZIP-files.

A ZIP archive can represent more or less the full bandwidth of file system properties, therefore reporting on archive members is implemented similar to \texttt{iter\_dir()}. The iterator produces an \texttt{ZipfileItem} instance with standard information on file system elements, such as size, or \texttt{mtime}.

**Parameters**

- \texttt{path (Path)} -- Path of the ZIP archive to report content for (iterate over).
- \texttt{fp (bool, optional)} -- If True, each file-type item includes a file-like object to access the file's content. This file handle will be closed automatically when the next item is yielded or the function returns.

**Yields**

\texttt{ZipfileItem}
dataloader_next.iter_collections.utils

Utilities and types for collection iterators

class datalad_next.iter_collections.utils.FileSystemItem

Bases: PathBasedItem, TypedItem

fp: IO | None = None

classmethod from_path(path: Path, *, link_target: bool = True)

Populate item properties from a single stat and readlink call

The given path must exist. The link_target flag indicates whether to report the result of readlink for a symlink-type path.

gid: int | None = None

link_target: PurePath | None = None

mode: int | None = None

mtime: float | None = None

size: int

type: FileSystemItemType

uid: int | None = None

class datalad_next.iter_collections.utils.FileSystemItemType(value)

Bases: Enum

Enumeration of file system path types

The associated str values are chosen to be appropriate for downstream use (e.g., as type labels in DataLad result records).

directory = 'directory'

file = 'file'

hardlink = 'hardlink'

specialfile = 'specialfile'

symlink = 'symlink'

class datalad_next.iter_collections.utils.NamedItem(name: 'Any')

Bases: object

name: Any

class datalad_next.iter_collections.utils.PathBasedItem(name: 'PurePath')

Bases: NamedItem
name: PurePath

class datalad_next.iter_collections.utils.TypedItem(type: 'Any')
    Bases: object
    type: Any

datalad_next.iter_collections.utils.compute_multihash_from_fp(fp, hash: List[str], bufsize=65536)
    Compute multiple hashes from a file-like

**datalad_next.runners**

Execution of subprocesses
This module import all relevant components for subprocess execution.

**datalad_next.tests.fixtures**

Collection of fixtures for facilitation test implementations

datalad_next.tests.fixtures.check_gitconfig_global()
    No test must modify a user's global Git config.
    If such modifications are needed, a custom configuration setup limited to the scope of the test requiring it must
    be arranged.

datalad_next.tests.fixtures.check_plaintext_keyring()
    No test must modify a user's keyring.
    If such modifications are needed, a custom keyring setup limited to the scope of the test requiring it must be
    arranged. The tmp_keyring fixture can be employed in such cases.

datalad_next.tests.fixtures.credman(datalad_cfg, tmp_keyring)
    Provides a temporary credential manager
    It comes with a temporary global datalad config and a temporary keyring as well.
    This manager can be used to deploy or manipulate credentials within the scope of a single test.

datalad_next.tests.fixtures.datalad_cfg()
    Temporarily alter configuration to use a plain "global" configuration
    The global configuration manager at datalad.cfg is reloaded after adjusting GIT_CONFIG_GLOBAL to point to
    a new temporary .gitconfig file.
    After test execution the file is removed, and the global ConfigManager is reloaded once more.
    Any test using this fixture will be skipped for Git versions earlier than 2.32, because the GIT_CONFIG_GLOBAL
    environment variable used here was only introduced with that version.

datalad_next.tests.fixtures.datalad_interactive_ui(monkeypatch)
    Yields a UI replacement to query for operations and stage responses
    No output will be written to STDOUT/ERR by this UI.
    A standard usage pattern is to stage one or more responses, run the to-be-tested code, and verify that the desired
    user interaction took place:
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

```python
> datalad_interactive_ui.staged_responses.append('skip')
> ...
> assert ... datalad_interactive_ui.log
```

datalad_next.tests.fixtures.datalad_noninteractive_ui

Yields a UI replacement to query for operations

No output will be written to STDOUT/ERR by this UI.

A standard usage pattern is to run the to-be-tested code, and verify that the desired user messaging took place:

```python
> ...
> assert ... datalad_interactive_ui.log
```

datalad_next.tests.fixtures.dataset

Provides a Dataset instance for a not-yet-existing repository

The instance points to an existing temporary path, but create() has not been called on it yet.

datalad_next.tests.fixtures.existing_dataset

Provides a Dataset instance pointing to an existing dataset/repo

This fixture uses an instance provided by the dataset fixture and calls create() on it, before it yields the Dataset instance.

datalad_next.tests.fixtures.existing_noannex_dataset

just like existing_dataset, but created with annex=False

datalad_next.tests.fixtures.http_credential

Provided an HTTP server, serving a temporary directory

The fixtures yields an instance of HTTPPath, providing the following essential attributes:

- path: Path instance of the served temporary directory
- url: HTTP URL to access the HTTP server

Server access requires HTTP Basic authentication with the credential provided by the webdav_credential fixture.

datalad_next.tests.fixtures.http_server

Like http_server but requiring authenticating with http_credential

datalad_next.tests.fixtures.httpbin

Does the same thing as httpbin_service, but skips on function-scope

httpbin_service always returns access URLs for HTTPBIN. However, in some cases it is simply not desirable to run a test. For example, the appveyor workers are more or less constantly unable to access the public service. This fixture is evaluated at function-scope and raises SkipTest whenever any of these undesired conditions is detected. Otherwise it just relays httpbin_service.

datalad_next.tests.fixtures.httpbin_service

Return canonical access URLs for the HTTPBIN service

This fixture tries to spin up a httpbin Docker container at localhost:8765; if successful, it returns this URL as the 'standard' URL. If the attempt fails, a URL pointing to the canonical instance is returned.

2.3. Python tooling
For tests that need to have the service served via a specific protocol (https vs http), the corresponding URLs are returned too. They always point to the canonical deployment, as some tests require both protocols simultaneously and a local deployment generally won’t have https.

```python
datalad_next.tests.fixtures.tmp_keyring()
```

Patch plaintext keyring to temporarily use a different storage

No credential read or write actions will impact any existing credential store of any configured backend.

The patched backend is yielded by the fixture.

```python
datalad_next.tests.fixtures.webdav_credential()
```

Provides a WebDAV server, serving a temporary directory

The fixtures yields an instance of `WebDAVPath`, providing the following essential attributes:

- **path**: Path instance of the served temporary directory
- **url**: HTTP URL to access the WebDAV server

Server access requires HTTP Basic authentication with the credential provided by the `webdav_credential` fixture.

### datalad_next.types

Custom types and dataclasses

<table>
<thead>
<tr>
<th>annexkey</th>
<th>git-annex key representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>archivist</td>
<td>dl+archive: archive member locator</td>
</tr>
<tr>
<td>enums</td>
<td>Type ENUMs</td>
</tr>
</tbody>
</table>

### datalad_next.types.annexkey

**git-annex key representation**

**class** `datalad_next.types.annexkey.AnnexKey` *(name: str, backend: str, size: int | None = None, mtime: int | None = None, chunksize: int | None = None, chunknumber: int | None = None)*

Bases: `object`

Representation of a git-annex key

https://git-annex.branchable.com/internals/key_format/

```python
backend: str
chunknumber: int | None = None
chunksize: int | None = None
```

**class method** `from_str` *(key: str)*

Return an `AnnexKey` instance from a key string

```python
mtime: int | None = None
```
name: str
	size: int | None = None

datajad_next.types.archivist
dl+archive: archive member locator
class datalad_next.types.archivist.ArchivistLocator(akey: AnnexKey, member: PurePosixPath, size: int, atype: ArchiveType | None = None)

Bases: object

Representation of a dl+archive: archive member locator

These locators are used by the datalad-archives and archivist git-annex special remotes. They identify a member of a archive that is itself identified by an annex key.

Each member is annotated with its size (in bytes). Optionally, the file format type of the archive can be annotated too.

Syntax of dl+archives: locators

The locators the following minimal form:

dl+archive:<archive-key>#path=<path-in-archive>

where <archive-key> is a regular git-annex key of an archive file, and <path-in-archive> is a POSIX-style relative path pointing to a member within the archive.

Two optional, additional attributes size and atype are recognized (only size is also understood by the datalad-archives special remote).

size declares the size of the (extracted) archive member in bytes:

dl+archive:<archive-key>#path=<path-in-archive>&size=<size-in-bytes>

atype declares the type of the containing archive using a label. Currently recognized labels are tar (a TAR archive, compressed or not), and zip (a ZIP archive). See ArchiveType for all recognized labels.

If no type information is given, ArchivistLocator.from_str() will try to determine the archive type from the archive key (via *E-type git-annex backends, such as DataLad’s default MD5E).

The order in the fragment part of the URL (after #) is significant. path must come first, followed by size or atype. If both size and atype are present, size must be declared first. A complete example of a URL is:

dl+archive:MD5-s389--e9f624eb778e6f945771c543b6e9c7b2#path=dir/file.csv&size=234&
→atype=tar

akey: AnnexKey

atype: ArchiveType | None = None

classmethod from_str(url: str)

Return ArchivistLocator from str form

member: PurePosixPath

size: int
**datalad_next.types.enums**

Type ENUMs

```python
class datalad_next.types.enums.ArchiveType(value)
    Bases: Enum
    Enumeration of archive types
    Each one should have an associated ArchiveOperations handler.

    tar = 'tar'
    zip = 'zip'
```

**datalad_next.uis**

UI abstractions for user communication

This module imports all necessary components.

**datalad_next.url_operations**

Handlers for operations on various URL types and protocols

Available handlers:

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>any</code></td>
<td>Meta URL handler with automatic scheme-based switching of implementations</td>
</tr>
<tr>
<td><code>file</code></td>
<td>Handler for operations, such as &quot;download&quot;, on file:// URLs</td>
</tr>
<tr>
<td><code>http</code></td>
<td>Handler for operations, such as &quot;download&quot;, on http(s):// URLs</td>
</tr>
<tr>
<td><code>ssh</code></td>
<td>Handler for operations, such as &quot;download&quot;, on ssh:// URLs</td>
</tr>
</tbody>
</table>

**datalad_next.url_operations.any**

Meta URL handler with automatic scheme-based switching of implementations

```python
class datalad_next.url_operations.any.AnyUrlOperations(cfg: ConfigManager | None = None)
    Bases: UrlOperations
    Handler for operations on any supported URLs

    The methods inspect a given URL and call the corresponding methods for the UrlOperations implementation that matches the URL best. The "best match" is the match expression of a registered URL handler that yields the longest match against the given URL.

    Parameter identity and semantics are unchanged with respect to the underlying implementations. See their documentation for details.

    An instance retains and reuses URL scheme handler instances for subsequent operations, such that held connections or cached credentials can be reused efficiently.
```
delete(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict

Call *UrlOperations.delete() for the respective URL scheme

download(from_url: str, to_path: Path | None, *, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) → Dict

Call *UrlOperations.download() for the respective URL scheme

is_supported_url(url) → bool

stat(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict

Call *UrlOperations.stat() for the respective URL scheme

upload(from_path: Path | None, to_url: str, *, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) → Dict

Call *UrlOperations.upload() for the respective URL scheme

datalad_next.url_operations.file

Handler for operations, such as "download", on file:// URLs

class datalad_next.url_operations.file.FileUrlOperations(*, cfg: ConfigManager | None = None)

Bases: UrlOperations

Handler for operations on file:// URLs

Access to local data via file-scheme URLs is supported with the same API and feature set as other URL-schemes (simultaneous content hashing and progress reporting.

delete(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict

Delete the target of a file:// URL

The target can be a file or a directory. If it is a directory, it has to be empty.

See datalad_next.url_operations.UrlOperations.delete() for parameter documentation and exception behavior.

Raises

UrlOperationsResourceUnknown -- For deletion targets found absent.

download(from_url: str, to_path: Path | None, *, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) → Dict

Copy a file:// URL target to a local path

See datalad_next.url_operations.UrlOperations.download() for parameter documentation and exception behavior.

Raises

UrlOperationsResourceUnknown -- For download targets found absent.

stat(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict

Gather information on a URL target, without downloading it

See datalad_next.url_operations.UrlOperations.stat() for parameter documentation and exception behavior.

Raises

UrlOperationsResourceUnknown -- For access targets found absent.
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

upload(from_path: Path | None, to_url: str | None, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) → Dict

Copy a local file to a file:// URL target

Any missing parent directories of the URL target are created as necessary.

See `datalad_next.url_operations.UrlOperations.upload()` for parameter documentation and exception behavior.

Raises

`FileNotFoundException` -- If the source file cannot be found.

datalad_next.url_operations.http

Handler for operations, such as "download", on http(s):// URLs

class datalad_next.url_operations.http.HttpUrlOperations(cfg=None, headers: Dict | None = None)
Bases: UrlOperations

Handler for operations on http(s):// URLs

This handler is built on the `requests` package. For authentication, it employs `datalad_next.utils.requests_auth.DataladAuth`, an adaptor that consults the DataLad credential system in order to fulfill HTTP authentication challenges.

download(from_url: str, to_path: Path | None, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) → Dict

Download via HTTP GET request

See `datalad_next.url_operations.UrlOperations.download()` for parameter documentation and exception behavior.

Raises

`UrlOperationsResourceUnknown` -- For download targets found absent.

get_headers(headers: Dict | None = None) → Dict

probe_url(url, timeout=10.0, headers=None)

Probe a HTTP(S) URL for redirects and authentication needs

This function performs a HEAD request against the given URL, while waiting at most for the given timeout duration for a server response.

Parameters

- `url` (str) -- URL to probe
- `timeout` (float, optional) -- Maximum time to wait for a server response to the probe
- `headers` (dict, optional) -- Any custom headers to use for the probe request. If none are provided, or the provided headers contain no 'user-agent' field, the default DataLad user agent is added automatically.

Returns

The first value is the URL against the final request was performed, after following any redirects and applying normalizations.

The second value is a mapping with a particular set of properties inferred from probing the webserver. The following key-value pairs are supported:
• 'is_redirect' (bool), True if any redirection occurred. This boolean property is a more accurate test than comparing input and output URL.

• 'status_code' (int), HTTP response code (of the final request in case of redirection).

• 'auth' (dict), present if the final server response contained any 'WWW-Authenticate' headers, typically the case for 401 responses. The dict contains a mapping of server-reported authentication scheme names (e.g., 'basic', 'bearer') to their respective properties (dict). These can be any nature and number, depending on the respective authentication scheme. Most notably, they may contain a 'realm' property that can be used to determine suitable credentials for authentication.

Return type
str or None, dict

Raises
requests.RequestException -- May raise any exception of the requests package, most notably ConnectionError, Timeout, TooManyRedirects, etc.

stat(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict
Gather information on a URL target, without downloading it
See datalad_next.url_operations.UrlOperations.stat() for parameter documentation and exception behavior.

Raises
UrlOperationsResourceUnknown -- For access targets found absent.

datalad_next.url_operations.ssh

Handler for operations, such as "download", on ssh:// URLs

class datalad_next.url_operations.ssh.SshUrlOperations(*, cfg: ConfigManager | None = None)
Bases: UrlOperations

Handler for operations on ssh:// URLs

For downloading files, only servers that support execution of the commands 'printf', 'ls -nl', 'awk', and 'cat' are supported. This includes a wide range of operating systems, including devices that provide these commands via the 'busybox' software.

Note: The present implementation does not support SSH connection multiplexing, (re-)authentication is performed for each request. This limitation is likely to be removed in the future, and connection multiplexing supported where possible (non-Windows platforms).

download(from_url: str, to_path: Path | None, *, credential: str | None = None, hash: str | None = None, timeout: float | None = None) → Dict
Download a file by streaming it through an SSH connection.

On the server-side, the file size is determined and sent. Afterwards the file content is sent via cat to the SSH client.

See datalad_next.url_operations.UrlOperations.download() for parameter documentation and exception behavior.

2.3. Python tooling
stat(...)
Gather information on a URL target, without downloading it
See `datalad_next.url_operations.UrlOperations.stat()` for parameter documentation and exception behavior.

upload(...)
Upload a file by streaming it through an SSH connection.
It, more or less, runs `ssh <host> 'cat > <path>'`.
See `datalad_next.url_operations.UrlOperations.upload()` for parameter documentation and exception behavior.

class datalad_next.url_operations.UrlOperations(*, cfg=ConfigManager | None = None)
Abstraction for operations on URLs
Support for specific URL schemes can be implemented via sub-classes. Such classes must comply with the following conditions:

- Any configuration look-up must be performed with the `self.cfg` property, which is guaranteed to be a `ConfigManager` instance.
- When downloads are to be supported, implement the `download()` method and comply with the behavior described in its documentation.

This class provides a range of helper methods to aid computation of hashes and progress reporting.

property cfg: ConfigManager

delete(...)
Delete a resource identified by a URL
Parameters
- `url` (str) -- Valid URL with any scheme supported by a particular implementation.
- `credential` (str, optional) -- The name of a dedicated credential to be used for authentication in order to perform the deletion. Particular implementations may or may not require or support authentication. They also may or may not support automatic credential lookup.
- `timeout` (float, optional) -- If given, specifies a timeout in seconds. If the operation is not completed within this time, it will raise a `TimeoutError`-exception. If timeout is None, the operation will never timeout.

Returns
A mapping of property names to values for the deletion.

Return type
dict

Raises
- `UrlOperationsRemoteError` -- This exception is raised on any deletion-related error on the remote side, with a summary of the underlying issues as its message. It may carry a status code (e.g. HTTP status code) as its `status_code` property. Any underlying exception must be linked via the `__cause__` property (e.g. `raise UrlOperationsRemoteError(...) from ...`).
• **UrlOperationsInteractionError** --

• **UrlOperationsAuthenticationError** --

• **UrlOperationsAuthorizationError** --

• **UrlOperationsResourceUnknown** -- Implementations that can distinguish several remote error types beyond indication a general `UrlOperationsRemoteError`: `UrlOperationsInteractionError` general issues in communicating with the remote side; `UrlOperationsAuthenticationError` for errors related to (failed) authentication at the remote; `UrlOperationsAuthorizationError` for (lack of) authorizing to access a particular resource or perform a particular operation; `UrlOperationsResourceUnknown` if the target of an operation does not exist.

• **TimeoutError** -- If `timeout` is given and the operation does not complete within the number of seconds that a specified by `timeout`.

```python
def download(from_url: str, to_path: Path | None = None, credential: str | None = None, hash: list[str] | None = None, timeout: float | None = None) -> Dict
```

Download from a URL to a local file or stream to stdout

**Parameters**

• **from_url** *(str)* -- Valid URL with any scheme supported by a particular implementation.

• **to_path** *(Path or None)* -- A local platform-native path or None. If None the downloaded data is written to `stdout`, otherwise it is written to a file at the given path. The path is assumed to not exist. Any existing file will be overwritten.

• **credential** *(str, optional)* -- The name of a dedicated credential to be used for authentication in order to perform the download. Particular implementations may or may not require or support authentication. They also may or may not support automatic credential lookup.

• **hash** *(list(algorithm_names), optional)* -- If given, must be a list of hash algorithm names supported by the `hashlib` module. A corresponding hash will be computed simultaneous to the download (without reading the data twice), and included in the return value.

• **timeout** *(float, optional)* -- If given, specifies a timeout in seconds. If the operation is not completed within this time, it will raise a `TimeoutError`-exception. If timeout is None, the operation will never timeout.

**Returns**

A mapping of property names to values for the completed download. If `hash` algorithm names are provided, a corresponding key for each algorithm is included in this mapping, with the hexdigest of the corresponding checksum as the value.

**Return type**

dict

**Raises**

• **UrlOperationsRemoteError** -- This exception is raised on any deletion-related error on the remote side, with a summary of the underlying issues as its message. It may carry a status code (e.g. HTTP status code) as its `status_code` property. Any underlying exception must be linked via the `__cause__` property (e.g. `raise UrlOperationsRemoteError(...) from ...`).

• **UrlOperationsInteractionError** --

• **UrlOperationsAuthenticationError** --
• **UrlOperationsAuthorizationError** --

• **UrlOperationsResourceUnknown** -- Implementations that can distinguish several remote error types beyond indication a general UrlOperationsRemoteError: UrlOperationsInteractionError general issues in communicating with the remote side; UrlOperationsAuthenticationError for errors related to (failed) authentication at the remote; UrlOperationsAuthorizationError for (lack of) authorizating to access a particular resource of perform a particular operation; UrlOperationsResourceUnknown if the target of an operation does not exist.

• **TimeoutError** -- If timeout is given and the operation does not complete within the number of seconds that a specified by timeout.

```
stat(url: str, *, credential: str | None = None, timeout: float | None = None) → Dict
```

Gather information on a URL target, without downloading it

**Returns**
A mapping of property names to values of the URL target. The particular composition of properties depends on the specific URL. A standard property is 'content-length', indicating the size of a download.

**Return type**
dict

**Raises**

• **UrlOperationsRemoteError** -- This exception is raised on any access-related error on the remote side, with a summary of the underlying issues as its message. It may carry a status code (e.g. HTTP status code) as its status_code property. Any underlying exception must be linked via the __cause__ property (e.g. raise UrlOperationsRemoteError(...) from ...).

• **UrlOperationsInteractionError** --

• **UrlOperationsAuthenticationError** --

• **UrlOperationsAuthorizationError** --

• **UrlOperationsResourceUnknown** -- Implementations that can distinguish several remote error types beyond indication a general UrlOperationsRemoteError: UrlOperationsInteractionError general issues in communicating with the remote side; UrlOperationsAuthenticationError for errors related to (failed) authentication at the remote; UrlOperationsAuthorizationError for (lack of) authorizating to access a particular resource of perform a particular operation; UrlOperationsResourceUnknown if the target of an operation does not exist.

• **TimeoutError** -- If timeout is given and the operation does not complete within the number of seconds that a specified by timeout.

```
upload(from_path: Path | None = None, to_url: str, *, credential: str | None = None, hash: list[| None = None, timeout: float | None = None) → Dict
```

Upload from a local file or stream to a URL

**Parameters**

• **from_path** (*Path or None*) -- A local platform-native path or None. If None the upload data is read from stdin, otherwise it is read from a file at the given path.

• **to_url** (*str*) -- Valid URL with any scheme supported by a particular implementation. The target is assumed to not conflict with existing content, and may be overwritten.
• **credential** (*str, optional*) -- The name of a dedicated credential to be used for authentication in order to perform the upload. Particular implementations may or may not require or support authentication. They also may or may not support automatic credential lookup.

• **hash** (*list(algorithm_names), optional*) -- If given, must be a list of hash algorithm names supported by the *hashlib* module. A corresponding hash will be computed simultaneous to the upload (without reading the data twice), and included in the return value.

• **timeout** (*float, optional*) -- If given, specifies a timeout in seconds. If the operation is not completed within this time, it will raise a *TimeoutError*-exception. If timeout is None, the operation will never timeout.

**Returns**
A mapping of property names to values for the completed upload. If hash algorithm names are provided, a corresponding key for each algorithm is included in this mapping, with the hexdigest of the corresponding checksum as the value.

**Return type**
dict

**Raises**

• **FileNotFoundException** -- If the source file cannot be found.

• **UrlOperationsRemoteError** -- This exception is raised on any deletion-related error on the remote side, with a summary of the underlying issues as its message. It may carry a status code (e.g. HTTP status code) as its *status_code* property. Any underlying exception must be linked via the *__cause__* property (e.g. raise *UrlOperationsRemoteError*(...) from ...).

• **UrlOperationsInteractionError** --

• **UrlOperationsAuthenticationError** --

• **UrlOperationsAuthorizationError** --

• **UrlOperationsResourceUnknown** -- Implementations that can distinguish several remote error types beyond indication a general *UrlOperationsRemoteError*: *UrlOperationsInteractionError* general issues in communicating with the remote side; *UrlOperationsAuthenticationError* for errors related to (failed) authentication at the remote; *UrlOperationsAuthorizationError* for (lack of) authorizing to access a particular resource of perform a particular operation; *UrlOperationsResourceUnknown* if the target of an operation does not exist.

• **TimeoutError** -- If timeout is given and the operation does not complete within the number of seconds that a specified by timeout.

```python
exception datalad_next.url_operations.UrlOperationsAuthenticationError(url: str, credential: dict | None = None, message: str | None = None, status_code: Any = None)
```

Bases: *UrlOperationsInteractionError*

```python
exception datalad_next.url_operations.UrlOperationsAuthorizationError(url: str, credential: dict | None = None, message: str | None = None, status_code: Any | None = None)
```

2.3. Python tooling
Bases: `UrlOperationsRemoteError`

```python
exception datalad_next.url_operations.UrlOperationsInteractionError(url, message=None, status_code: Any | None = None)
```

```
Bases: `UrlOperationsRemoteError`

exception datalad_next.url_operations.UrlOperationsRemoteError(url, message=None, status_code: Any | None = None)
```

```
Bases: Exception

property message

property status_code

property url
```

```
exception datalad_next.url_operations.UrlOperationsResourceUnknown(url, message=None, status_code: Any | None = None)
```

```
Bases: `UrlOperationsRemoteError`

A connection request succeeded in principle, but target was not found

Equivalent of an HTTP404 response.
```

datalad_next.utils

Assorted utility functions

```python
class datalad_next.utils.ParamDictator(params: Dict)
```

```
Bases: object

Parameter dict access helper

This class can be used to wrap a dict containing function parameter name-value mapping, and get/set values by parameter name attribute rather than via the `__getitem__` dict API.
```

datalad_next.utils.get_specialremote_credential_envpatch(remote_type, cred)

Create an environment path for a particular remote type and credential

**Returns**

A dict with all required items to patch the environment, or None if not enough information is available, or nothing needs to be patched.

**Return type**

dict or None

```
datalad_next.utils.get_specialremote_credential_properties(params)
```

Determine properties of credentials special remote configuration

The input is a parameterization as it would be given to `git annex initremote|enableremote <name> ...`, or as stored in `remote.log`. These parameters are inspected and a dictionary of credential properties, suitable for `CredentialManager.query()` is returned. This inspection may involve network activity, e.g. HTTP requests.

**Parameters**

```python
params (list or dict) -- Either a list of strings of the format 'param=value', or a dictionary with parameter names as keys.
```
Returns
Credential property name-value mapping. This mapping can be passed to CredentialManager.query(). If no credential properties could be inferred, for example, because the special remote type is not recognized None is returned.

Return type
dict or None
datalad_next.utils.get_specialremote_param_dict(params)

Parameters
params (list) --

Return type
dict
datalad_next.utils.needs_specialremote_credential_envpatch(remote_type)

Returns whether the environment needs to be patched with credentials

Returns
False, if the special remote type is not recognized as one needing credentials, or if there are credentials already present. True, otherwise.

Return type
bool
datalad_next.utils.update_specialremote_credential(srtype, credman, credname, credprops, credtype_hint=None, duplicate_hint=None)

Parameters
• srtype (str) --
• credman (CredentialManager) --
• credname (str or Name) --
• credprops (dict) --

2.4 Git remote helpers

2.4.1 Git-remote helpers

| datalad_annex         | git-remote-datalad-annextofetch/pushviaanygit-annex special remote |
**datalad_next.gitremotes.datalad_annex**

git-remote-datalad-annex to fetch/push via any git-annex special remote

In essence, this Git remote helper bootstraps a utility repository in order to push/fetch the state of a repository to any location accessible by any git-annex special remote implementation. All information necessary for this bootstrapping is taken from the remote URL specification. The internal utility repository is removed again after every invocation. Therefore changes to the remote access configuration can be made any time by simply modifying the configured remote URL.

When installed, this remote helper is invoked for any "URLs" that start with the prefix `datalad-annex::`. Following this prefix, two types of specifications are support.

1. Plain parameters list:
   
   `datalad-annex::?type=<special-remote-type>&[...][exporttree=yes]`

   In this case the prefix is followed by a URL query string that comprises all necessary (and optional) parameters that would be normally given to the `git annex initremote` command. It is required to specify the special remote type, and it is possible to request "export" mode for any special remote that supports it. Depending on the chosen special remote additional parameters may be required or supported. Please consult the git-annex documentation at https://git-annex.branchable.com/special_remotes/

2. URL:
   
   `datalad-annex::<url>[?...]`

   Alternatively, an actual URL can be given after the prefix. In this case, the, now optional, URL query string can still be used to specify arbitrary parameters for special remote initialization. In addition, the query string specification can use Python-format-style placeholder to reference particular URL components as parameters values, in order to avoid double-specification.

   The list of supported placeholders is `scheme`, `netloc`, `path`, `fragment`, `username`, `password`, `hostname`, `port`, corresponding to the respective URL components. In addition, a `noquery` placeholder is supported, which resolves to the entire URL except any query string. An example of such a URL specification is:

   `datalad-annex::file:///tmp/example?type=directory&directory={path}&encryption=none`

   which would initialize a `type=directory` special remote pointing at `/tmp/example`.

Caution with collaborative workflows

There is no protection against simultaneous, conflicting repository state uploads from two different locations! Similar to git-annex’s "export" feature, this feature is most appropriately used as a dataset deposition mechanism, where uploads are conducted from a single site only -- deposited for consumption by any number of parties.

If this Git remote helper is to be used for multi-way collaboration, with two or more parties contributing updates, it is advisable to employ a separate `datalad-annex::` target site for each contributor, such that only one site is pushing to any given location. Updates are exchanged by the remaining contributors adding the respective other `datalad-annex::` sites as additional Git remotes, analog to forks of a repository.

Special remote type support

In addition to the regular list of special remotes, plain http(s) access via URLs is also supported via the 'web' special remote. For such cases, only the base URL and the 'type=web' parameter needs to be given, e.g:

```
git clone 'datalad-annex::https://example.com?type=web&url={noquery}'
```
When a plain URL is given, with no parameter specification in a query string, the parameters type=web and exporttree=yes are added automatically by default. This means that this remote helper can clone from any remote deposit accessible via http(s) that matches the layout depicted in the next section.

Remote layout

The representation of a repository at a remote depends on the chosen type of special remote. In general, two files will be deposited. One text file containing a list of Git refs contained in the deposit, and one ZIP file with a (compressed) archive of a bare Git repository. Beside the idiosyncrasies of particular special remotes, to major modes determine the layout of a remote deposit. In "normal" mode, two annex keys (XDLRA--refs, XDLRA--repo-export) will be deposited. In "export" mode, a directory tree is created that is designed to blend with arbitrary repository content, such that a git remote and a git-annex export can be pushed to the same location without conflicting with each other. The aforementioned files will be represented like this:

```
.datalad
  └ dotgit # named to not be confused with an actual Git repository
    └ refs
    └ repo.zip
```

The default LZMA-compression of the ZIP file (in both export and normal mode) can be turned off with the dladotgit=uncompressed URL parameter.

Credential handling

Some git-annex special remotes require the specification of credentials via environment variables. With the URL parameter dlacredential=<name> it is possible to query DataLad for a user/password credential to be used for this purpose. This convenience functionality is supported for the special remotes glacier, s3, and webdav.

When a credential of the given name does not exist, or no credential name was specified, an attempt is made to determine a suitable credential based on, for example, a detected HTTP authentication realm. If no matching credential could be found, the user will be prompted to enter a credential. After having successfully established access, the entered credential will be saved in the local credential store.

DataLad-based credentials are only utilized, when the native git-annex credential setup via environment variables is not in use (see the documentation of a particular special remote implementation for more information).

Implementation details

This Git remote implementation uses two extra repositories, besides the repository (R) it is used with, to do its work:

(A) A tiny repository that is entirely bootstrapped from the remote URL, and is used to retrieve/deposit a complete state of the actual repo an a remote site, via a git-annex special remote setup.

(B) A local, fully functional mirror repo of the remotely stored repository state.

On fetch/push the existence of both additional repositories is ensured. The remote state of retrieved via repo (A), and unpacked to repo (B). The actual fetch/push Git operations are performed locally between the repo (R) and repo (B). On push, repo (B) is then packed up again, and deposited on the remote site via git-annex transfer in repo (A).

Due to a limitation of this implementation, it is possible that when the last upload step fails, Git nevertheless advances the pushed refs, making it appear as if the push was completely successful. That being said, Git will still issue a message (error: failed to push some refs to..) and the git-push process will also exit with a non-zero status. In addition, all of the remote's refs will be annotated with an additional ref named refs/dlra-upload-failed/<remote-name>/<ref-name> to indicate the upload failure. These markers will be automatically removed after the next successful upload.

Note: Confirmed to work with git-annex version 8.20211123 onwards.

2.4. Git remote helpers 73
Todo:

• At the moment, only one format for repository deposition is supported (a ZIP archive of a working bare repository). However this is not a good format for the purpose of long-term archiving, because it require a functional Git installation to work with. It would be fairly doable to make the deposited format configurable, and support additional formats. An interesting one would be a fast-export stream, basically a plain text serialization of an entire repository.

• recognize that a different repo is being pushed over an existing one at the remote

• think about adding additional information into the header of refs maybe give it some kind of stamp that also makes it easier to validate by the XDLRA backend

• think about preventing duplication between the repo and its local mirror could they safely share git objects? If so, in which direction?

class datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote(gitdir, remote, url, instream=<_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8'>, outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>, errstream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>)

Bases: object
git-remote-helper implementation

communicate() is the entrypoint.

communicate()

Implement the necessary pieces of the git-remote-helper protocol

Uses the input, output and error streams configured for the class instance.

get_mirror_refs()

Return the refs of the current mirror repo

Return type
str

get_remote_refs()

Report remote refs

The underlying special remote is asked whether it has the key containing the refs list for the remote. If it does, it is retrieved and reported.

Returns

If the remote has refs, they are returned as a string, formatted like a refs file in a Git directory. Otherwise, None is returned.

Return type
str or None

internal_parameters = ('dladotgit=uncompressed', 'dlacredential=')
log(*args, level=2)
    Send log messages to the errstream

property mirrorrepo
    Local remote mirror repository
    If accessed when there is no local mirror repo, as new one is created automatically, either from the remote state (if there is any), or an empty one.

    Returns
    This is always only a plain Git repository (bare).

    Return type
    GitRepo

refs_key = 'XDLRA--refs'

replace_mirrorrepo_from_remote_deposit()
    Replaces the local mirror repo with one obtained from the remote
    This method assumes that the remote does have one. This should be checked by inspecting get_remote_refs() before calling this method.

replace_mirrorrepo_from_remote_deposit_if_needed()
    Replace the mirror if the remote has refs and they differ

    Parameters
    mirror.refs (str, optional) -- If given, must be formatted like get_mirror_refs() would do.

replace_remote_deposit_from_mirrorrepo()
    Package the local mirror repo, up, and copy to the special remote
    The mirror is assumed to be ready/complete. It will be cleaned with gc to minimize the upload size. The mirrorrepo is then compressed into an LZMA ZIP archive, and a separate refs list for it is created in addition. Both are then copied to the special remote.

repo_export_key = 'XDLRA--repo-export'

property repoannex
    Repo annex repository
    If accessed when there is no repo annex, as new one is created automatically. It is bootstrapped entirely from the parameters encoded in the remote URL.

    Returns
    This is always an annex repository. It is configured with a single special remote, parameterized from the Git repo URL.

    Return type
    AnnexRepo

    Raises
    • CommandLineError
    • ValueError

    safe_content = ['branches', 'hooks', 'info', 'objects', 'refs', 'config', 'packed-refs', 'description', 'HEAD']
send(msg)
Communicate with Git

support_githelper_options = {'verbosity': EnsureInt()}

xdlra_key_locations = {'XDLRA--refs': {'loc': '.datalad/dotgit/refs', 'prefix': '3f7/4a3'}, 'XDLRA--repo-export': {'loc': '.datalad/dotgit/repo.zip', 'prefix': 'eb3/ca0'}}

2.5 Git-annex backends

2.5.1 Git-annex backends

<table>
<thead>
<tr>
<th>base</th>
<th>Interface and essential utilities to implement external git-annex backends</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdlra</td>
<td>git-annex external backend XDLRA for git-remote-datalad-annex</td>
</tr>
</tbody>
</table>

datalad_next.annexbackends.base

Interface and essential utilities to implement external git-annex backends

exception datalad_next.annexbackends.base.AnnexError

Bases: Exception

Common base class for all annexbackend exceptions.

class datalad_next.annexbackends.base.Backend(annex)

Bases: object

Metaclass for backends.

It implements the communication with git-annex via the external backend protocol. More information on the protocol is available at https://git-annex.branchable.com/design/external_backend_protocol/

External backends can be built by implementing the abstract methods defined in this class.

annex

The Master object to which this backend is linked. Master acts as an abstraction layer for git-annex.

Type

Master

abstract can_verify()

Returns whether the backend can verify the content of files match a key it generated. The verification does not need to be cryptographically secure, but should catch data corruption.

Return type

bool

error(error_msg)

Communicate a generic error.

Can be sent at any time if things get too messed up to continue. If the program receives an error() from git-annex, it can exit with its own error(). Eg.: self.annex.error("Error received. Exiting.") raise SystemExit
Parameters

```
error_msg (str) -- The error message received from git-annex
```

**abstract gen_key(local_file)**

Examine the content of `local_file` and from it generate a key.

While it is doing this, it can send any number of PROGRESS messages indicating the position in the file
that it's gotten to.

Parameters

```
local_file (str) -- Path for which to generate a key. Note that in some cases, local_file
may contain whitespace.
```

Returns

The generated key.

Return type

`str`

Raises

```
BackendError -- If the file could not be received from the backend.
```

**abstract is_cryptographically_secure()**

Returns whether keys it generates are verified using a cryptographically secure hash.

Note that sha1 is not a cryptographically secure hash any longer. A program can change its answer to this
question as the state of the art advances, and should aim to stay ahead of the state of the art by a reasonable
amount of time.

Return type

`bool`

**abstract is_stable()**

Returns whether a key it has generated will always have the same content. The answer to this is almost
always yes; URL keys are an example of a type of key that may have different content at different times.

Return type

`bool`

**abstract verify_content(key, content_file)**

Examine a file and verify it has the content expected given a key

While it is doing this, it can send any number of PROGRESS messages indicating the position in the file
that it's gotten to.

If `can_verify() == False`, git-annex not ask to do this.

Return type

`bool`

**exception datalad_next.annexbackends.base.BackendError**

Bases: `AnnexError`

Must be raised by the backend when a request did not succeed.

**class datalad_next.annexbackends.base.Master**(output=<<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>)

Bases: `object`

Metaclass for backends.
input
Where to listen for git-annex request messages. Default: sys.stdin
  Type
  io.TextIOBase

output
Where to send replies and backend messages Default: sys.stdout
  Type
  io.TextIOBase

backend
A class implementing the Backend interface to which this master is linked.
  Type
  Backend

LinkBackend(backend)
Link the Master to a backend. This must be done before calling Listen()

Parameters
  backend (Backend) -- A class implementing Backend interface to which this master will be linked.

Listen(input=<_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8'>)
Listen on input for messages from git annex.

Parameters
  input (io.TextIOBase) -- Where to listen for git-annex request messages. Default: sys.stdin

Raises
  NotLinkedError -- If there is no backend linked to this master.

debug(*args)
Tells git-annex to display the message if --debug is enabled.

Parameters
  message (str) -- The message to be displayed to the user

error(*args)
Generic error. Can be sent at any time if things get too messed up to continue. When possible, raise a BackendError inside the respective functions. The backend program should exit after sending this, as git-annex will not talk to it any further.

Parameters
  error_msg (str) -- The error message to be sent to git-annex

progress(progress)
Indicates the current progress of the transfer (in bytes). May be repeated any number of times during the transfer process, but it's wasteful to update the progress until at least another 1% of the file has been sent. This is highly recommended for *_store(). (It is optional but good for *_retrieve().)

Parameters
  progress (int) -- The current progress of the transfer in bytes.

exception datalad_next.annexbackends.base.NotLinkedError
  Bases: AnnexError

Will be raised when a Master instance is accessed without being linked to a Backend instance.
class datalad_next.annexbackends.base.Protocol(backend)
    Bases: object
    Helper class handling the receiving part of the protocol (git-annex to backend) It parses the requests coming from git-annex and calls the respective method of the backend object.

    command(line)

    do_CANVERIFY()

    do_ERROR(message)

    do_GENKEY(*arg)

    do_GETVERSION()

    do_ISCRYPTOGRAPHICALLYSECURE()

    do_ISSTABLE()

    do_VERIFYKEYCONTENT(*arg)

    lookupMethod(command)

exception datalad_next.annexbackends.base.ProtocolError
    Bases: AnnexError
    Base class for protocol errors

exception datalad_next.annexbackends.base.UnexpectedMessage
    Bases: ProtocolError
    Raised when git-annex sends a message which is not expected at the moment

exception datalad_next.annexbackends.base.UnsupportedRequest
    Bases: ProtocolError
    Must be raised when an optional request is not supported by the backend.

datalad_next.annexbackends.xdlra

git-annex external backend XDLRA for git-remote-datalad-annex

class datalad_next.annexbackends.xdlra.DataladRepoAnnexBackend(annex)
    Bases: Backend
    Implementation of an external git-annex backend

    This backend is tightly coupled to the git-remote-datalad-annex and hardly of any general utility. It is essentially aiming to be the leanest possible implementation to get git-annex to transport the content of two distinct files to and from a special remote. This backend is unlike most backends, because there is no fixed association of a particular file content to a particular key. In other words, the key content is expected to change without any change in the key name.

    Only two keys are supported:
    • XDLRA--refs
    • XDLRA--repo-export

    XDLRA--refs contains a “refs” list of a Git repository, similar to the output of `git for-each-ref`. XDLRA--repo-export hold a ZIP archive of a bare Git repository.

2.5. Git-annex backends
can_verify()

Returns whether the backend can verify the content of files match a key it generated. The verification does not need to be cryptographically secure, but should catch data corruption.

Return type
bool

gen_key(local_file)

Examine the content of local_file and from it generate a key.

While it is doing this, it can send any number of PROGRESS messages indication the position in the file that it's gotten to.

Parameters
local_file (str) -- Path for which to generate a key. Note that in some cases, local_file may contain whitespace.

Returns
The generated key.

Return type
str

Raises
BackendError -- If the file could not be received from the backend.

is_cryptographically_secure()

Returns whether keys it generates are verified using a cryptographically secure hash.

Note that sha1 is not a cryptographically secure hash any longer. A program can change its answer to this question as the state of the art advances, and should aim to stay ahead of the state of the art by a reasonable amount of time.

Return type
bool

is_stable()

Returns whether a key it has generated will always have the same content. The answer to this is almost always yes; URL keys are an example of a type of key that may have different content at different times.

Return type
bool

verify_content(key, content_file)

Examine a file and verify it has the content expected given a key

While it is doing this, it can send any number of PROGRESS messages indicating the position in the file that it's gotten to.

If can_verify() == False, git-annex not ask to do this.

Return type
bool

datalad_next.annexbackends.xdlra.main()

Entry point for the backend utility
2.6 Git-annex special remotes

2.6.1 Git-annex special remotes

<table>
<thead>
<tr>
<th>SpecialRemote(annex)</th>
<th>Base class of all datalad-next git-annex special remotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>archivist</td>
<td>git-annex special remote <em>archivist</em> for obtaining files from archives</td>
</tr>
<tr>
<td>uncurl</td>
<td>uncurl git-annex external special remote</td>
</tr>
</tbody>
</table>

**datalad_next.annexremotes.SpecialRemote**

```python
class datalad_next.annexremotes.SpecialRemote(annex):
    Bases: SpecialRemote
    Base class of all datalad-next git-annex special remotes

    get_remote_gitcfg(remotetypename: str, name: str, default: Any | None = None, **kwargs)
    Get a particular Git configuration item for the special remote
    This target configuration here is not the git-annex native special remote configuration that is provided or altered with *initremote* and *enableremote*, and is committed to the *git-annex* branch. Instead this is a clone and remote specific configuration, declared in Git’s configuration system.
    The configuration items queried have the naming scheme:
    
    remote.<remotename>.<remotetypename>-<name>
    datalad.<remotetypename>.<name>
    
    where <remotename> is the name of the Git remote, the special remote is operating under, <remotetypename> is the name of the special remote implementation (e.g., *uncurl*), and <name> is the name of a particular configuration flavor.
    
    Parameters
    • remotetypename (str) -- Name of the special remote implementation configuration is requested for.
    • name (str) -- The name of the "naked" configuration item, without any sub/sections. Must be a valid git-config variable name, i.e., case-insensitive, only alphanumeric characters and -, and must start with an alphabetic character.
    • default -- A default value to be returned if there is no configuration.
    • **kwargs -- Passed on to datalad_next.config.ConfigManager.get()

    Returns
    If a remote-specific configuration exists, it is reported. Otherwise a remote-type specific configuration is reported, or the default provided with the method call, if no configuration is found at all.

    Return type
    Any

    property remotename: str
    Name of the (git) remote the special remote is operating under
```

2.6. Git-annex special remotes
property repo: LeanAnnexRepo

Returns a representation of the underlying git-annex repository

An instance of LeanAnnexRepo is returned, which intentionally provides a restricted API only. In order to
limit further proliferation of the AnnexRepo API.

datalad_next.annexremotes.archivist

git-annex special remote archivist for obtaining files from archives

class datalad_next.annexremotes.archivist.ArchivistRemote(annex)

Bases: SpecialRemote
git-annex special remote archivist for obtaining files from archives

Successor of the datalad-archive special remote. It claims and acts on particular archive locator "URLs", regis-
tered for individual annex keys (see datalad_next.types.archivist.ArchivistLocator). These locators
identify another annex key that represents an archive (e.g., a tarball or a zip files) that contains the respective
annex key as a member. This special remote trigger the extraction of such members from any candidate archive
when retrieval of a key is requested.

This special remote cannot store or remove content. The desired usage is to register a locator "URL" for any
relevant key via git annex addurl|registerurl or datalad addurls.

Configuration

The behavior of this special remote can be tuned via a number of configuration settings.

datalad.archivist.legacy-mode=yes[no]

If enabled, all special remote operations fall back onto the legacy datalad-archives special remote
implementation. This mode is only provided for backward-compatibility. This legacy implementation un-
conditionally downloads archive files completely, and keeps an internal cache of the full extracted archive
around. The implied 200% (or more) storage cost overhead for obtaining a complete dataset can be pro-
hibitive for datasets tracking large amount of data (in archive files).

Implementation details

CHECKPRESENT

When performing a non-download test for the (continued) presence of an annex key (as triggered via git annex
fsck --fast or git annex checkpresentkey), the underlying archive containing a key will NOT be in-
spected. Instead, only the continued availability of the annex key for the containing archive will be tested. In
other words: this implementation trust the archive member annotation to be correct/valid, and it also trusts the
archive content to be unchanged. The latter will be generally the case, but may no with URL-style keys.

Not implementing such a trust-approach would have a number of consequences. Depending on where the archive
is located (local/remote) and what format it is (fsspec-inspectable or not), we would need to download it com-
pletely in order to verify a matching archive member. Moreover, an archive might also reference another archive
as a source, leading to a multiplication of transfer demands.

checkpresent(key: str) → bool

Verifies continued availability of the archive referenced by the key

No content verification of the archive, or of the particular archive member is performed. See "Implementation
details" of this class for a rational.
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

Returns
True if the referenced archive key is present on any remote. False if not.

Return type
bool

checkurl(url: str) → bool
Parses ArchivistLocator-style URLs
Returns True for any syntactically correct URL with all required properties.
The implementation is identical to claimurl().

claimurl(url: str) → bool
Returns True for ArchivistLocator-style URLs
Only a lexical check is performed. Any other URL will result in False to be returned.

initremote()
This method does nothing, because the special remote requires no particular setup.

prepare()
Prepare the special remote for requests by git-annex
If the special remote is instructed to run in "legacy mode", all subsequent operations will be processed by the datalad-archives special remote implementation!

remove(key: str)
Raises UnsupportedRequest. This operation is not supported.

transfer_retrieve(key: str, localfilename: str)
Retrieve an archive member from a (remote) archive
All registered locators for a requested key will be sorted by availability and size of the references archives. For each archive the most suitable handler will be initialized, and extraction of the identified member will be attempted. If that fails, the next handler is tried until all candidate handlers are exhausted. Depending on the archive availability and type, archives may need to be retrieved from remote sources.

transfer_store(key: str, filename: str)
Raises UnsupportedRequest. This operation is not supported.

datalad_next.annexremotes.archivist.main()
CLI entry point installed as git-annex-remote-archivist

datalad_next.annexremotes.uncurl

uncurl git-annex external special remote

This implementation is a git-annex accessible interface to datalad-next's URL operations framework. It serves two main purposes:

1. Combine git-annex's capabilities of registering and accessing file content via URLs with DataLad's access credential management and (additional or alternative) transport protocol implementations.

2. Minimize the maintenance effort for datasets (primarily) composed from content that is remotely accessible via URLs from systems other than Datalad or git-annex in the event of an infrastructure transition (e.g. moving to a different technical system or a different data organization on a storage system).
Requirements

This special remote implementation requires git-annex version 8.20210127 (or later) to be available.

Download helper

The simplest way to use this remote is to initialize it without any particular configuration:

```
$ git annex initremote uncurl type=external externaltype=uncurl encryption=none
initremote uncurl ok
(recording state in git...)
```

Once initialized, or later enabled in a clone, `git-annex addurl` will check with the `uncurl` remote whether it can handle a particular URL, and will let the remote perform the download in case of positive response. By default, the remote will claim any URLs with a scheme that the local datalad-next installation supports. This always includes `file://`, `http://`, and `https://`, but is extensible, and a particular installation may also support `ssh://` (by default when openssh is installed), or other schemes.

This additional URL support is also available for other commands. Here is an example how `datalad addurls` can be given any uncurl-supported URLs (here an SSH-URL) directly, provided that the uncurl remote was initialized for a dataset (as shown above):

```
$ echo '[["url":"ssh://my.server.org/home/me/file", "file":"dummy"\}'] \       | datalad addurls - '{url}' '{file}'
```

This makes legacy commands (e.g., `datalad download-url`), unnecessary, and facilitates the use of more advanced `datalad addurls` features (e.g., automatic creation of subdatasets) that are not provided by lower-level commands like `git annex addurl`.

Download helper with credential management support

With this setup, download requests now also use DataLad's credential system for authentication. DataLad will automatically lookup matching credentials, prompt for manual entry if none are found, and offer to store them securely for later use after having used them successfully:

```
$ git annex addurl http://httpbin.org/basic-auth/myuser/mypassword
Credential needed for access to http://httpbin.org/basic-auth/myuser/mypassword
user: myuser
password:
password (repeat):
Enter a name to save the credential
(for accessing http://httpbin.org/basic-auth/myuser/mypassword) securely for future re-use, or 'skip' to not save the credential
name: httpbin-dummy

addurl http://httpbin.org/basic-auth/myuser/mypassword (from uncurl) (to ...) ok
(recording state in git...)
```

By adding files via downloads from URLs in this fashion, datasets can be built that track information across a range of locations/services, using a possibly heterogeneous set of access methods.
This feature is very similar to the `datalad` special remote implementation included in the core DataLad package. The difference here is that alternative implementations of downloaoders are employed and the `datalad-next` credential system is used instead of the "providers" mechanism from DataLad's core package.

**Transforming recorded URLs**

The main benefit of using `uncurl` is, however, only revealed when the original snapshot of where data used to be accessible becomes invalid, maybe because data were moved to a different storage system, or simply a different host.

This would typically require an update of each, now broken, access URL. For datasets with thousands or even millions of files this can be an expensive operation. For data portal operators providing a large number of datasets it is even more tedious.

`uncurl` enables programmatic, on-access URL rewriting. This is similar, in spirit, to Git's `url.<base>.insteadOf` URL modification feature. However, modification possibilities reach substantially beyond replacing a base URL.

This feature is based on two customizable settings: 1) a URL template; and 2) a set of match expressions that extract additional identifiers from any recorded access URL for an annex key.

Here is an example: Let's say a file in a dataset has a recorded access URL of:

```plaintext
https://data.example.org/c542/s7612_figure1.pdf
```

We can let `uncurl` know that c542 is actually an identifier for a particular collection of items in this data store. Likewise s7612 is an identifier of a particular item in that collection, and `figure1.pdf` is the name of a component in that collection item. The following Python regular expression can be used to "decompose" the above URL into these semantic components:

```plaintext
(?P<site>https://[^/]+)/(?P<collection>c[^/]+)/(?P<item>s[^/]+)_({component}.*$)
```

This expression is not the most readable, but it basically chunks the URL into segments of `(?P<name>...)`, so-called named groups (see a live demo of this expression).

This expression, and additional ones like it, can set as a configuration parameter of an `uncurl` remote setup. Extending the configuration established by the `initremote` call above:

```bash
$ git annex enableremote uncurl \
   'match=(?P<site>https://[^/]+)/(?P<collection>c[^/]+)/(?P<item>s[^/]+)_(?P<component>...).*)$
```

The last argument is quoted to prevent it from being processed by the shell.

With the match expression configured, URL rewriting can be enabled by declaring a URL template as another configuration item. The URL template uses the Python Format String Syntax. If the new URL for the file above is now `http://newsite.net/ex-archive/c542_s7612_figure1.pdf`, we can declare the following URL template to have `uncurl` go to the new site:

```plaintext
http://newsite.net/ex-archive/{collection}_{item}_{component}
```

This template references the identifiers of the named groups we defined in the match expression. Again, the URL template can be set via `git annex enableremote`:

```bash
$ git annex enableremote uncurl \ 
   'url=http://newsite.net/ex-archive/{collection}_{item}_{component}'
```

There is no need to separate the `enableremote` calls. Both configuration can be given at the same time. In fact, they can also be given to `initremote` immediately.
The three identifiers site, collection, item, and component are actually a custom addition to a standard set of identifiers that are available for composing URLs via a template.

- **datalad_dsid** - the DataLad dataset ID (UUID)
- **annex_dirhash** - "mixed" variant of the two level hash for a particular key (uses POSIX directory separators, and included a trailing separator)
- **annex_dirhash_lower** - "lower case" variant of the two level hash for a particular key (uses POSIX directory separators, and included a trailing separator)
- **annex_key** - git-annex key name for a request
- **annex_remoteuuid** - UUID of the special remote (location) used by git-annex
- **git_remotename** - Name of the Git remote for the uncurl special remote

**Note:** The URL template must "resolve" to a complete and valid URL. This cannot be verified at configuration time, because even the URL scheme could be a dynamic setting.

### Uploading content

The uncurl special remote can upload file content or store annex keys via supported URL schemes whenever a URL template is defined. At minimum, storing at file:// and ssh:// URLs are supported. But other URL scheme handlers with upload support may be available at a local DataLad installation.

### Deleting content

As for uploading, deleting content is only permitted with a configured URL template. Moreover, it also depends on the delete operation being supported for a particular URL scheme.

### Configuration overrides

Both match expressions and the URL template can also be configured in a dataset's configuration (committed branch configuration, or any Git configuration scope (local, global, system) using the following configuration item names:

- remote.<remotename>.uncurl-url
- remote.<remotename>.uncurl-match

where <remotename> is the name of the special remote in the dataset.

A URL template provided via configuration overrides one defined in the special remote setup via init/enableremote.

Match expressions defined as configuration items extend the set of match expressions that may be included in the special remote setup via init/enableremote. The remote.<remotename>.uncurl-match configuration item can be set as often as necessary (which one match expression each).
Tips

When multiple match expressions are defined, it is recommended to use unique names for each match-group to avoid collisions.

class datalad_next.annexremotes.uncurl.UncurlRemote(annex)
Bases: SpecialRemote

checkpresent(key)
Requests the remote to check if a key is present in it.

Parameters
key (str) --

Returns
True if the key is present in the remote. False if the key is not present.

Return type
bool

Raises
RemoteError -- If the presence of the key couldn’t be determined, e.g. in case of connection error.

checkurl(url)
When running git-annex addurl, this is called after CLAIMURL indicated that we could handle a URL. It can return information on the URL target (e.g., size of the download, a target filename, or a sequence thereof with additional URLs pointing to individual components that would jointly make up the full download from the given URL. However, all of that is optional, and a simple True returned is sufficient to make git-annex call TRANSFER RETRIEVE.

claimurl(url)
Needs to check if want to handle a given URL

If match expressions are configured, matches the URL against all known URL expressions, and returns True if there is any match, or False otherwise.

If no match expressions are configured, return True of the URL scheme is supported, or False otherwise.

extract_tmpl_props(tmpl, *, urls=None, key=None)

get_key_urls(key) → list[str]

get_mangled_url(fallback_url, tmpl, tmpl_props)

initremote()
Gets called when git annex initremote or git annex enableremote are run. This is where any one-time setup tasks can be done, for example creating the remote folder. Note: This may be run repeatedly over time, as a remote is initialized in different repositories, or as the configuration of a remote is changed. So any one-time setup tasks should be done idempotently.

Raises
RemoteError -- If the remote could not be initialized.

is_recognized_url(url)

prepare()
Tells the remote that it’s time to prepare itself to be used. Gets called whenever git annex is about to access any of the below methods, so it shouldn’t be too expensive. Otherwise it will slow down operations like git annex whereis or git annex info.
Internet connection can be established here, though it's recommended to defer this until it's actually needed.

**Raises**

RemoteError -- If the remote could not be prepared.

**remove**(key)

Requests the remote to remove a key's contents.

**Parameters**

key (str) --

**Raises**

RemoteError -- If the key couldn't be deleted from the remote.

**transfer_retrieve**(key, filename)

Get the file identified by key from the remote and store it in local_file.

While the transfer is running, the remote can repeatedly call annex.progress(size) to indicate the number of bytes already stored. This will influence the progress shown to the user.

**Parameters**

- key (str) -- The Key to get from the remote.
- local_file (str) -- Path where to store the file. Note that in some cases, local_file may contain whitespace.

**Raises**

RemoteError -- If the file could not be received from the remote.

**transfer_store**(key, filename)

Store the file in local_file to a unique location derived from key.

It's important that, while a Key is being stored, checkpresent(key) not indicate it's present until all the data has been transferred. While the transfer is running, the remote can repeatedly call annex.progress(size) to indicate the number of bytes already stored. This will influence the progress shown to the user.

**Parameters**

- key (str) -- The Key to be stored in the remote. In most cases, this is going to be the remote file name. It should be at least be unambiguously derived from it.
- local_file (str) -- Path to the file to upload. Note that in some cases, local_file may contain whitespace. Note that local_file should not influence the filename used on the remote.

**Raises**

RemoteError -- If the file could not be stored to the remote.
## 2.7 DataLad patches

Patches that are automatically applied to DataLad when loading the `datalad-next` extension package.

### 2.7.1 DataLad patches

<table>
<thead>
<tr>
<th><code>annexrepo</code></th>
<th>Credential support for <code>AnnexRepo.enable_remote()</code> and siblings enable</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>common_cfg</code></td>
<td>Change the default of <code>datalad.annex.retry</code> to 1</td>
</tr>
<tr>
<td><code>configuration</code></td>
<td>Enable <code>configuration()</code> to query global scope without a dataset</td>
</tr>
<tr>
<td><code>create_sibling_ghlike</code></td>
<td>Improved credential handling for <code>create_sibling_&lt;github-like&gt;()</code></td>
</tr>
<tr>
<td><code>customremotes_main</code></td>
<td>Connect <code>log_progress</code>-style progress reporting to git-annex</td>
</tr>
<tr>
<td><code>distribution_dataset</code></td>
<td>DatasetParameter support for <code>resolve_path()</code></td>
</tr>
<tr>
<td><code>interface_utils</code></td>
<td>Uniform pre-execution parameter validation for commands</td>
</tr>
<tr>
<td><code>push_optimize</code></td>
<td>Make push avoid refspec handling for special remote push targets</td>
</tr>
<tr>
<td><code>push_to_export_remote</code></td>
<td>Add support for export to WebDAV remotes to <code>push()</code></td>
</tr>
<tr>
<td><code>test_keyring</code></td>
<td>Recognize <code>DATALAD_TESTS_TMP_KEYRING_PATH</code> to set alternative secret storage</td>
</tr>
<tr>
<td><code>siblings</code></td>
<td>Auto-deploy credentials when enabling special remotes</td>
</tr>
</tbody>
</table>

**`datalad_next.patches.annexrepo`**

Credential support for `AnnexRepo.enable_remote()` and siblings enable

Supported targets for automatic credential deployments are determined by `needs_specialremote_credential_envpatch()`. At the time of this writing this includes the git-annex built-in remote types `webdav`, `s3`, and `glacier`.

This patch also changes the function to raise its custom exception with the context of an original underlying exception for better error reporting.

```
`datalad_next.patches.annexrepo.annexRepo__enable_remote`(self, name, options=None, env=None)
```

Enables use of an existing special remote

#### Parameters

- `name` *(str)* -- name, the special remote was created with
- `options` *(list, optional)* --
**datalad_next.patches.common_cfg**

Change the default of `datalad.annex.retry` to 1

This prevents unconditional retries, and thereby improves the legibility of errors (now only one error instead of three identical errors).

This change does not override user-settings, only the default.

**datalad_next.patches.configuration**

Enable `configuration()` to query global scope without a dataset

```python
class datalad_next.patches.configuration.Configuration
    Bases: Configuration
datalad_next.patches.configuration.configuration(action, scope, specs, res_kwargs, ds=None)
```

**datalad_next.patches.create_sibling_ghlike**

Improved credential handling for `create_sibling_<github-like>()`

This patch makes the storage of a newly entered credential conditional on a successful authorization, in the spirit of datalad/datalad#3126.

Moreover, stored credentials now contain a `realm` property that identified the API endpoint. This makes it possible to identify candidates of suitable credentials without having to specific their name, similar to a request context url used by the old providers setup.

This automatic realm-based credential lookup is now also implemented. When no credential name is specified, the most recently used credential matching the API realm will be used automatically. If determined like this, it will be tested for successful authorization, and will then be stored again with an updated `last-used` timestamp.

**datalad_next.patches.customremotes_main**

Connect `log_progress`-style progress reporting to git-annex

This patch introduces a dedicated progress log handler as a proxy between standard datalad progress logging and a git-annex special remote as an approach to report (data transfer) progress to a git-annex parent process.

This functionality is only (to be) used in dedicated special remote processes.

```python
class datalad_next.patches.customremotes_main.AnnexProgressLogHandler(annexremote: SpecialRemote)
```

Bases: Handler

Log handler to funnel progress logs to git-annex

For this purpose the handler wraps `datalad_next.annexremotes.SpecialRemote` instance. When it receives progress log messages, it converts any increment reports to absolute values, and then calls the special remote's `send_progress()` method, which will cause the respective progress update protocol message to be issued.

**Note:** Git-annex only supports "context-free" progress reporting. When a progress report is send, it is assumed to be on a currently running transfer. Only a single integer value can be reported, and it corresponds to the number of bytes transferred.
This approach implemented here cannot distinguish progress reports that correspond to git-annex triggered
data transfers and other (potentially co-occurring) operations. The likelihood of unrelated operations reporting
growth is relatively low, because this handler is only supposed to be used in dedicated special remote processes,
but remains possible.

This implementation is set up to support tracking multiple processes, and could report one of them selectively.
However, at present any progress update is relayed to git-annex directly. This could lead to confusing and non-
linear progress reporting.

```python
emit(record: LogRecord)
```

Process a log record

Any incoming log record, compliant with http://docs.datalad.org/design/progress_reporting.html is pro-
cessed. Increment reports are converted to absolute values, and each update is eventually passed on to
special remote, which issues a progress report to git-annex.

datalad_next.patches.customremotes_main.only_progress_logrecords(record: LogRecord) → bool

Log filter to ignore any non-progress log message

datalad_next.patches.customremotes_main.patched_underscore_main(args: list, cls: Type[SpecialRemote])

Full replacement for datalad.customremotes.main._main()

Its only purpose is to create a running instance of a SpecialRemote. The only difference to the original in datalad-
core is that once this instance exists, it is linked to a log handler that converts incoming progress log messages
to the equivalent annex protocol progress reports.

This additional log handler is a strict addition to the log handling setup established at this point. There should
be no interference with any other log message processing.

See also:

AnnexProgressLogHandler

datalad_next.patches.distribution_dataset

DatasetParameter support for resolve_path()

This is the standard result of EnsureDataset, which unlike the datalad-core version actually carries a Dataset in-
stance.

This patch ensure the traditional handling of "dataset instance from a string-type parameter in this context.

datalad_next.patches.distribution_dataset.resolve_path(path, ds=None, ds_resolved=None)

Resolve a path specification (against a Dataset location)

Any path is returned as an absolute path. If, and only if, a dataset object instance is given as ds, relative paths are
interpreted as relative to the given dataset. In all other cases, relative paths are treated as relative to the current
working directory.

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

Parameters

2.7. DataLad patches
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

- **path** *(str or PathLike or list)* -- Platform-specific path specific path specification. Multiple path specifications can be given as a list

- **ds** *(Dataset or PathLike or None)* -- Dataset instance to resolve relative paths against.

- **ds_resolved** *(Dataset or None)* -- A dataset instance that was created from *ds* outside can be provided to avoid multiple instantiation on repeated calls.

**Returns**

When a list was given as input a list is returned, a Path instance otherwise.

**Return type**

`pathlib.Path` object or list(Path)

datalad_next.patches.interface_utils

Uniform pre-execution parameter validation for commands

With this patch commands can now opt-in to receive fully validated parameters. This can substantially simplify the implementation complexity of a command at the expense of a more elaborate specification of the structural and semantic properties of the parameters.

For details on implementing validation for individual commands see `datalad_next.commands. ValidatedInterface`.

datalad_next.patches.interface_utils.get_allargs_as_kwargs*(call, args, kwargs)*

Generate a kwargs dict from a call signature and *args, **kwargs

Basically resolving the argnames for all positional arguments, and resolving the defaults for all kwargs that are not given in a kwargs dict

**Returns**

The first return value is a mapping of argument names to their respective values. The second return value in the tuple is a set of argument names for which the effective value is identical to the default declared in the signature of the callable. The third value is a set with names of all mandatory arguments, whether or not they are included in the returned mapping.

**Return type**

(dict, set, set)

datalad_next.patches.push_optimize

Make push avoid refspec handling for special remote push targets

This change introduces a replacement for core’s `push.py:_push()` with a more intelligible flow. It replaces the stalled https://github.com/datalad/datalad/pull/6666

Importantly, it makes one behavior change, which is desirable IMHO. Instead of rejecting to git-push any refspec for a repo with a detached HEAD, it will attempt to push a git-annex branch for an AnnexRepo. The respective test that ensured this behavior beyond the particular conditions the original problem occurred in was adjusted accordingly.

All push tests from core are imported and executed to ensure proper functioning.

Summary of the original commits patching the core implementation:

- Consolidate publication dependency handling in one place
- Consolidate tracking of git-push-dryrun exec Make a failed attempt discriminable from no prior attempt.
- Factor out helper to determine refspeccs-to-push for a target
• Consolidate more handling of git-pushed and make conditional on an actual git-remote target. This change is breaking behavior, because previously a source repository without an active branch would have been rejected for a push attempt. However, this is a bit questionable, because the git-annex branch might well need a push.

• Simplify push-logic: no need for a fetch, if there is no git-push

• Factor out helper to sync a remote annex-branch

• Adjust test to constrain the evaluated conditions (replacement tests is included here). As per the reasoning recorded in datalad#1811 (comment) the test ensuring the continue fix of datalad#1811 is actually verifying a situation that is not fully desirable. It prevents pushing of the 'git-annex' branch whenever a repo is on a detached HEAD. This change let's the test run on a plain Git repo, where there is indeed nothing to push in this case.

datalad_next.patches.push_to_export_remote

Add support for export to WebDAV remotes to `push()`

This approach generally works for any special remote configured with `exporttree=yes`, but is only tested for `type=webdav`. A smooth operation requires automatic deployment of credentials. Support for that is provide and limited by the capabilities of `needs_specialremote_credential_envpatch()`.

datalad_next.patches.push_to_export_remote.get_export_records(repo: AnnexRepo) → Generator

Read exports that git-annex recorded in its 'export.log'-file

Interpret the lines in export.log. Each line has the following structure:

time-stamp " " source-annex-uuid ":" destination-annex-uuid " " treeish

Parameters

repo (AnnexRepo) -- The annex repo from which exports should be determined

Returns

Generator yielding one dictionary for each export entry in git-annex. Each dictionary contains the keys: "timestamp", "source-annex-uuid", "destination-annex-uuid", "treeish". The timestamp-value is a float, all other values are strings.

Return type

Generator

datalad_next.patches.test_keyring

Recognize DATALAD_TESTS_TMP_KEYRING_PATH to set alternative secret storage

Within `pytest` DataLad uses the plaintext keyring backend. This backend has no built-in way to configure a custom file location for secret storage from the outside. This patch looks for a DATALAD_TESTS_TMP_KEYRING_PATH environment variable, and uses its value as a file path for the storage.

This makes it possible to (temporarily) switch storage. This feature is used by the `tmp_keyring` pytest fixture. This patch is needed in addition to the test fixture in order to apply such changes also to child processes, such as special remotes and git remotes.
**datalad_next.patches.siblings**

Auto-deploy credentials when enabling special remotes

This is the companion of the `annexRepo__enable_remote` patch, and simply removes the webdav-specific credential handling in `siblings()` . It is no longer needed, because credential deployment moved to a lower layer, covering more special remote types.

Manual credential entry on `enableremote` is not implemented here, but easily possible following the patterns from `datalad-annex:` and `create_sibling_webdav()`

## 2.8 Developer Guide

### 2.8.1 The developer's guide to datalad-next

This guide sheds light on new and reusable subsystems developed in `datalad-next`. The target audience are developers that intend to build up on or use functionality provided by this extension.

**datalad-next's Constraint System**

`datalad_next.constraints` implements a system to perform data validation, coercion, and parameter documentation for commands via a flexible set of "Constraints". You can find an overview of available Constraints in the respective module overview of the *Python tooling*.

**Adding parameter validation to a command**

In order to equip an existing or new command with the constraint system, the following steps are required:

- Set the commands base class to `ValidatedInterface`:

```python
from datalad_next.commands import ValidatedInterface

@build_doc
class MyCommand(ValidatedInterface):
    """Download from URLs""
```

- Declare a `_validator_` class member:

```python
from datalad_next.commands import (
    EnsureCommandParameterization,
    ValidatedInterface,
)

@build_doc
class MyCommand(ValidatedInterface):
    """Download from URLs""

    _validator_ = EnsureCommandParameterization(dict(
        [...]
    ))
```
• Determine for each parameter of the command whether it has constraints, and what those constraints are. If you're transitioning an existing command, remove any constraints= declaration in the _parameter_ class member.

• Add a fitting Constraint declaration for each parameter into the _validator_ as a key-value pair where the key is the parameter and its value is a Constraint. There does not need to be a Constraint per parameter; only add entries for parameters that need validation.

```python
from datalad_next.commands import (EnsureCommandParameterization,
                                   ValidatedInterface,
                                   )
from datalad_next.constraints import EnsureChoice
from datalad_next.constraints.dataset import EnsureDataset

@build_doc
class Download(ValidatedInterface):
    """Download from URLs""

    _validator_ = EnsureCommandParameterization(dict(
        dataset=EnsureDataset(installed=True),
        force=EnsureChoice('yes','no','maybe'),
    ))
```

**Combining constraints**

Constraints can be combined in different ways. The |, &, and () operators allow AND, OR, and grouping of Constraints. The following example from the download command defines a chain of possible Constraints:

```python
spec_item_constraint = url2path_constraint | (  
    EnsureJSON() | EnsureURLFilenamePairFromURL()  
) & url2path_constraint
```

Constraints can also be combined using AnyOf or AllOf MultiConstraints, which correspond almost entirely to | and &. Here's another example from the download command:

```python
spec_constraint = AnyOf(  
    spec_item_constraint,  
    EnsureListOf(spec_item_constraint),  
    EnsureGeneratorFromFileLike(  
        spec_item_constraint,  
        exc_mode='yield',  
    ),
)
```

One can combine an arbitrary number of Constraints. They are evaluated in the order in which they were specified. Logical OR constraints will return the value from the first constraint that does not raise an exception, and logical AND constraints pass the return values of each constraint into the next.
Implementing additional constraints

TODO

Parameter Documentation

TODO

Contributing to datalad-next

We're happy about contributions of any kind to this project - thanks for considering making one!

Please take a look at CONTRIBUTING.md for an overview of development principles and common questions, and get in touch in case of questions or to discuss features, bugs, or other issues.
INDICES AND TABLES

- genindex
- modindex
- search
PYTHON MODULE INDEX

datalad_next.annexbackends.base, 76
datalad_next.annexbackends.xd1ra, 79
datalad_next.annexremotes.archivist, 82
datalad_next.annexremotes.uncurl, 83
datalad_next.archive_operations, 31
datalad_next.archive_operations.tarfile, 31
datalad_next.commands, 33
datalad_next.config, 34
datalad_next.constraints, 34
datalad_next.constraints.base, 34
datalad_next.constraints.basic, 36
datalad_next.constraints.compound, 39
datalad_next.constraints.dataset, 44
datalad_next.constraints.exceptions, 45
datalad_next.constraintsformats, 41
datalad_next.constraints.git, 44
datalad_next.constraints.parameter, 42
datalad_next.credman, 47
datalad_next.credman.manager, 47
datalad_next.datasets, 52
datalad_next.exceptions, 52
datalad_next.gitremotes.datalad_annex, 72
datalad_next.iter_collections, 52
datalad_next.iter_collections.directory, 53
datalad_next.iter_collections.gitworktree, 53
datalad_next.iter_collections.tarfile, 55
datalad_next.iter_collections.utils, 57
datalad_next.iter_collections.zipfile, 56
datalad_next.patches.annexrepo, 89
datalad_next.patches.common_cfg, 90
datalad_next.patches.configuration, 90
datalad_next.patches.create_sibling_ghlike, 90
datalad_next.patches.customremotes_main, 90
datalad_next.patches.distribution_dataset, 91
datalad_next.patches.interface_utils, 92
datalad_next.patches.push_optimize, 92
datalad_next.patches.push_to_export_remote, 93
datalad_next.patches.siblings, 94
datalad_next.patches.test_keyring, 93
INDEX

A
akey (datalad_next.types.archivist.ArchivistLocator attribute), 61
AllOf (class in datalad_next.constraints.base), 34
annex (datalad_next.annexbackends.base.Backend attribute), 76
AnnexError, 76
AnnexProgressLogHandler (class in datalad_next.patches.customremotes_main), 90
annexRepo__enable_remote() (in module datalad_next.patches.annexrepo), 89
AnyOf (class in datalad_next.constraints.base), 35
AnyUrlOperations (class in datalad_next.url_operations.any), 62
ArchiveOperations (class in datalad_next.archive_operations), 32
ArchiveType (class in datalad_next.types.enums), 62
ArchivistLocator (class in datalad_next.types.archivist), 61
ArchivistRemote (class in datalad_next.annexremotes.archivist), 82
atype (datalad_next.types.archivist.ArchivistLocator attribute), 61

cfg (datalad_next.url_operations.UrlOperations property), 66
check_gitconfig_global() (in module datalad_next.tests.fixtures), 58
check_plaintext_keyring() (in module datalad_next.tests.fixtures), 58
checkpresent() (datalad_next.annexremotes.archivist.ArchivistRemote method), 82
ccheckpresent() (datalad_next.annexremotes.uncurl.UncurlRemote method), 87
checkurl() (datalad_next.annexremotes.archivist.ArchivistRemote method), 83
checkurl() (datalad_next.annexremotes.uncurl.UncurlRemote method), 87
chunknumber (datalad_next.types.annexkey.AnnexKey attribute), 60
chunksize (datalad_next.types.annexkey.AnnexKey attribute), 60
claimurl() (datalad_next.annexremotes.archivist.ArchivistRemote method), 80
claimurl() (datalad_next.annexremotes.uncurl.UncurlRemote method), 87
close() (datalad_next.archive_operations.ArchiveOperations method), 32
close() (datalad_next.archive_operations.tarfile.TarArchiveOperations method), 32
ccommand() (datalad_next.annexbackends.base.Protocol method), 79

CommandParametrizationError, 45
communicate() (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote method), 74
compute_multihash_from_fp() (in module datalad_next.iter_collections.utils), 58
Configuration (class in datalad_next.patches.configuration), 90
configuration() (in module datalad_next.patches.configuration), 90
Constraint (class in datalad_next.constraints.base), 35
constraint (datalad_next.constraints.compound.ConstraintWithPassthrough property), 66

B
Backend (class in datalad_next.annexbackends.base), 76
backend (datalad_next.annexbackends.base.Master attribute), 78
backend (datalad_next.types.annexkey.AnnexKey attribute), 60
BackendError, 77

C
can_verify() (datalad_next.annexbackends.base.Backend method), 76
can_verify() (datalad_next.annexbackends.xdla.DataladRepoAnnexBackend method), 79
causend_by (datalad_next.constraints.exceptions.ConstraintError property), 45
cfg (datalad_next.archive_operations.ArchiveOperations property), 32

101
Datalad Next, Release 1.0.0b3+25.g0c44b0.dirty

property), 39
constraint (datalad_next.constraints.composite.WithDescription
property), 40
datalad_next.constraints.exceptions.ConstraintError
property), 45
ConstraintError, 45
ConstraintErrors, 45
ConstraintWithPassthrough (class in datalad_next.constraints.composite), 39
datalad_next.constraints.exceptions.ConstraintError
property), 45
datalad_next.datasets
context (datalad_next.constraints.exceptions.ConstraintError
property), 45
datalad_next.exceptions
context_labels (datalad_next.constraints.exceptions.ParameterContextErrors
property), 47
credential_manager() (in module datalad_next.credman.manager), 47
credentials() (in module datalad_next.api), 8
credman() (in module datalad_next.tests.fixtures), 58

D
datalad_cfg() (in module datalad_next.tests.fixtures), 58
datalad_interactive_ui() (in module datalad_next.tests.fixtures), 58
datalad_next.annexbackends.base
module, 76
datalad_next.annexbackends.xdlra
module, 79
datalad_next.annexremotes.archivist
module, 82
datalad_next.annexremotes.uncurl
module, 83
datalad_next.archive_operations
module, 31
datalad_next.archive_operations.tarfile
module, 31
datalad_next.commands
module, 33
datalad_next.config
module, 34
datalad_next.constraints
module, 34
datalad_next.constraints.composite
module, 36
datalad_next.constraints.dataset
module, 44
datalad_next.constraints.exceptions
module, 45
datalad_next.constraints.formats
module, 41
datalad_next.constraints.git
module, 44
datalad_next.constraints.parameter
module, 42
datalad_next.credman
module, 47
datalad_next.credman.manager
module, 47
datalad_next.datasets
module, 52
datalad_next.exceptions
module, 52
datalad_next.gitremotes.datalad_annex
module, 72
datalad_next.iter_collections
module, 52
datalad_next.iter_collections.directory
module, 53
datalad_next.iter_collections.gitworktree
module, 53
datalad_next.iter_collections.tarfile
module, 55
datalad_next.iter_collections.utils
module, 57
datalad_next.iter_collections.zipfile
module, 56
datalad_next.patches.annexrepo
module, 89
datalad_next.patches.common_cfg
module, 90
datalad_next.patches.configuration
module, 90
datalad_next.patches.create_sibling_ghlike
module, 90
datalad_next.patches.customremotes_main
module, 90
datalad_next.patches.distribution_dataset
module, 91
datalad_next.patches.interface_utils
module, 92
datalad_next.patches.push_optimize
module, 92
datalad_next.patches.push_to_export_remote
module, 93
datalad_next.patches.siblings
module, 94
datalad_next.patches.test_keyring
module, 93
datalad_next.runners
module, 58
datalad_next.tests.fixtures
module, 58
datalad_next.types
module, 58
module, 60
datalad_next.types.annexkey
module, 60
datalad_next.types.archivist
module, 61
datalad_next.types.enums
module, 62
datalad_next.uis
module, 62
datalad_next.url_operations
module, 62
datalad_next.url_operations.any
module, 62
datalad_next.url_operations.file
module, 63
datalad_next.url_operations.http
module, 64
datalad_next.url_operations.ssh
module, 65
datalad_next.utils
module, 70
datalad_noninteractive_ui()
  (in module data-
  lad_next.tests.fixtures), 59
DataladRepoAnnexBackend
  (class in data-
  lad_next.annexbackends.xdlra), 79
dataset()
  (in module datalad_next.tests.fixtures), 59
DatasetParameter
  (class in data-
  lad_next.constraints.base), 36
debug()
  (datalad_next.annexbackends.base.Master
  method), 78
delete()
  (datalad_next.url_operations.any.AnyUrlOperations
  method), 62
delete()
  (datalad_next.url_operations.file.FileUrlOperations
  method), 63
delete()
  (datalad_next.url_operations.UrlOperations
  method), 66
description
  (datalad_next.constraints.exceptions.ParameterConstraintContext
  attribute), 46
directory
  (datalad_next.iter_collections.gitworktree.GitTreeItemType
  attribute), 53
directory
  (datalad_next.iter_collections.utils.FileSystemItemType
  attribute), 57
DirectoryItem
  (class in data-
  lad_next.iter_collections.directory), 53
do_CANVERIFY()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_ERROR()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_GENKEY()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_GETVERSION()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_ISCRYPTOGRAPHICALLYSECURE()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_ISSTABLE()
  (datalad_next.annexbackends.base.Protocol
  method), 79
do_VERIFYKEYCONTENT()
  (datalad_next.annexbackends.base.Protocol
  method), 79
download()
  (datalad_next.url_operations.any.AnyUrlOperations
  method), 63
download()
  (datalad_next.url_operations.file.FileUrlOperations
  method), 63
download()
  (datalad_next.url_operations.http.HttpUrlOperations
  method), 64
download()
  (datalad_next.url_operations.ssh.SshUrlOperations
  method), 65
download()
  (datalad_next.url_operations.UrlOperations
  method), 67
download()
  (in module datalad.api), 11
E
emit()
  (datalad_next.patches.customremotes_main.annexprogressloghandler.AnnexProgressLogHandler
  method), 91
EnsureBool
  (class in datalad_next.constraints.basic), 36
EnsureCallable
  (class in datalad_next.constraints.basic), 36
EnsureChoice
  (class in datalad_next.constraints.basic), 37
EnsureCommandParameterization
  (class in datalad_next.constraints.parameter), 42
EnsureDataset
  (class in datalad_next.constraints.dataset), 44
EnsureDType
  (class in datalad_next.constraints.basic), 37
EnsureFloat
  (class in datalad_next.constraints.basic), 39
EnsureGeneratorFromFileLike
  (class in datalad_next.constraints.compound), 39
EnsureGitRefName
  (class in datalad_next.constraints.git), 44
EnsureInt
  (class in datalad_next.constraints.basic), 37
EnsureIterableOf
  (class in datalad_next.constraints.compound), 39
EnsureJSON
  (class in datalad_next.constraints.formats), 41
EnsureKeyChoice
  (class in datalad_next.constraints.basic), 37
EnsureListOf
  (class in datalad_next.constraints.compound), 39
EnsureMapping
  (class in datalad_next.constraints.compound), 40
EnsureNone
  (class in datalad_next.constraints.basic), 37
get_specialremote_credential_properties() (in module datalad_next.utils), 70
get_specialremote_param_dict() (in module datalad_next.utils), 71
gid (datalad_next.iter_collections.utils.FileSystemItem attribute), 57
gitsha (datalad_next.iter_collections.gitworktree.GitWorktreeFilesystemItem attribute), 54
get_specialremote_credential_properties() (in module datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote attribute), 74
gitsha (datalad_next.iter_collections.gitworktree.GitWorktreeFileSystemItem attribute), 54
GitTreeItemType (class in datalad_next.iter_collections.gitworktree), 53
gittype (datalad_next.iter_collections.gitworktree.GitWorktreeFileSystemItem attribute), 54
gittype (datalad_next.iter_collections.gitworktree.GitWorktreeItem attribute), 54
GitWorktreeFileSystemItem (class in datalad_next.iter_collections.gitworktree), 54
GitWorktreeItem (class in datalad_next.iter_collections.gitworktree), 54
hardlink (datalad_next.iter_collections.utils.FileSystemItemType attribute), 57
http_credential() (in module datalad_next.tests.fixtures), 59
http_server() (in module datalad_next.tests.fixtures), 59
http_server() (in module datalad_next.iter_collections.tarfile), 59
httpbin() (in module datalad_next.iter_collections.tarfile), 59
httpbin_service() (in module datalad_next.iter_collections.tarfile), 59
HttpUrlOperations (class in datalad_next.url_operations.http), 64
initremote() (datalad_next.annexremotes.archivist.ArchivistRemote method), 83
initremote() (datalad_next.annexremotes.urc.UrlRemote method), 83
input (datalad_next.annexbackends.base.Master attribute), 77
input_description (datalad_next.constraints.base.Constraint property), 35
input_description (datalad_next.constraints.compound.WithDescription property), 40
input_synopsis (datalad_next.constraints.base.Constraint property), 35
input_synopsis (datalad_next.constraints.compound.WithDescription property), 40
is_cryptographically_secure() (datalad_next.annexbackends.base.Backend method), 77
is_cryptographically_secure() (datalad_next.annexbackends.xdlra.DataladRepoAnnexBackend method), 80
is_stable() (datalad_next.annexbackends.base.Backend method), 77
is_stable() (datalad_next.annexbackends.xdlra.DataladRepoAnnexBackend method), 80
is_supported_url() (datalad_next.url_operations.any.AnyUrlOperations method), 63
item_constraint (datalad_next.constraints.shuffle.EnsureIterableOf property), 39
items() (datalad_next.constraints.exceptions.ParameterContextErrors method), 47
iter_dir() (in module datalad_next.iter_collections.directory), 53
iter_gitworktree() (in module datalad_next.iter_collections.gitworktree), 54
iter_tar() (in module datalad_next.iter_collections.tarfile), 55
iter_zip() (in module datalad_next.iter_collections.zipfile), 56
joint_validation() (datalad_next.constraints.parameter.EnsureCommandParameterization method), 43
label (datalad_next.constraints.exceptions.ParameterConstraintContext property), 46
LeanAnnexRepo (class in datalad_next.datasets), 52
link_target (datalad_next.iter_collections.tarfile.TarfileItem attribute), 55
link_target (datalad_next.iter_collections.utils.FileSystemItem attribute), 57
LinkBackend() (datalad_next.annexbackends.base.Master method), 78
Listen() (datalad_next.annexbackends.base.Master method), 78
log() (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote method), 74

Index 105
Datalad Next, Release 1.0.0b3+25.g0cb44b0.dirty

long_description() (datalad_next.constraints.base.AllOf method), 34
long_description() (datalad_next.constraints.base.AnyOf method), 35
long_description() (datalad_next.constraints.base.Constraint method), 35
long_description() (datalad_next.constraints.basic.EnsureBool method), 36
long_description() (datalad_next.constraints.basic.EnsureCallable method), 36
long_description() (datalad_next.constraints.basic.EnsureChoice method), 36
long_description() (datalad_next.constraints.basic.EnsureDType method), 37
long_description() (datalad_next.constraints.basic.EnsureKeyChoice method), 37
long_description() (datalad_next.constraints.basic.EnsureRange method), 38
long_description() (datalad_next.constraints.basic.EnsureStr method), 38
long_description() (datalad_next.constraints.basic.EnsureStrPrefix method), 38
long_description() (datalad_next.constraints.basic.EnsureValue method), 38
long_description() (datalad_next.constraints.compound.ConstraintWithPassthrough method), 39
long_description() (datalad_next.constraints.compound.WithDescription method), 41
lookupMethod() (datalad_next.annexbackends.base.Protocol method), 79
ls_file_collection() (in module datalad.api), 13

M
main() (in module datalad_next.annexbackends.xdlra), 80
main() (in module datalad_next.annexremotes.archivist), 83
main() (in module datalad_next.annexremotes.uncurl), 88

Master (class in datalad_next.annexbackends.base), 77
member (datalad_next.types.archivist.ArchivistLocator attribute), 61
message (datalad_next.url_operations.UrlOperationsRemoteError property), 70
messages (datalad_next.constraints.exceptions.ParameterContextErrors property), 47
mirrorrepo (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote property), 75
mode (datalad_next.iter_collections.utils.FileSystemItem attribute), 57
module
datalad_next.annexbackends.base, 76
datalad_next.annexbackends.xdlra, 79
datalad_next.annexremotes.archivist, 82
datalad_next.annexremotes.uncurl, 83
datalad_next.archive_operations, 31
datalad_next.archive_operations.tarfile, 31
datalad_next.commands, 33
datalad_next.config, 34
datalad_next.constraints, 34
datalad_next.constraints.base, 34
datalad_next.constraints.basic, 36
datalad_next.constraints.compound, 39
datalad_next.constraints.dataset, 44
datalad_next.constraints.exceptions, 45
datalad_next.constraints.forms, 41
datalad_next.constraints.git, 44
datalad_next.constraints.parameter, 42
datalad_next.credman, 47
datalad_next.credman.manager, 47
datalad_next.datasets, 52
datalad_next.exceptions, 52
datalad_next.gitremotes.datalad_annex, 72
datalad_next.iter_collections, 52
datalad_next.iter_collections.directory, 53
datalad_next.iter_collections.gitworktree, 53
datalad_next.iter_collections.tarfile, 55
datalad_next.iter_collections.utils, 57
datalad_next.iter_collections.zipfile, 56
datalad_next.patches.annexrepo, 89
datalad_next.patches.common_cfg, 90
datalad_next.patches.configuration, 90
datalad_next.patches.create_sibling_ghlike, 90
datalad_next.patches.customremotes_main, 90
datalad_next.patches.distribution_dataset, 91
datalad_next.patches.interface_utils, 92
datalad_next.patches.push_optimize, 92
replace_mirrorrepo_from_remote_deposit_if_needed() (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote method), 75

replace_remote_deposit_from_mirrorrepo() (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote method), 75

repo (datalad_next.annexremotes.SpecialRemote property), 81

repo_export_key (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote attribute), 75

repoannex (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote property), 75

RepoAnnexGitRemote (class in datalad_next.gitremotes.datalad_annex), 74

resolve_path() (in module datalad_next.patches.distribution_dataset), 91

S

safe_content (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote attribute), 75

secret_names (datalad_next.credman.manager.CredentialManager attribute), 51

send() (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote method), 75

set() (datalad_next.credman.manager.CredentialManager method), 51

short_description() (datalad_next.constraints.base.AllOf method), 35

short_description() (datalad_next.constraints.base.AnyOf method), 35

short_description() (datalad_next.constraints.base.Constraint method), 36

short_description() (datalad_next.constraints.base.EnsureBool method), 36

short_description() (datalad_next.constraints.base.EnsureCallable method), 36

short_description() (datalad_next.constraints.base.EnsureChoice method), 36

short_description() (datalad_next.constraints.base.EnsureDType method), 37

short_description() (datalad_next.constraints.base.EnsureKeyChoice method), 37

short_description() (datalad_next.constraints.base.EnsurePath method), 38

short_description() (datalad_next.constraints.basic.EnsureRange method), 38

short_description() (datalad_next.constraints.basic.EnsureStr method), 38

short_description() (datalad_next.constraints.basic.EnsurePrefix method), 38

short_description() (datalad_next.constraints.basic.EnsureValue method), 38

short_description() (datalad_next.constraints.compound.ConstraintWithPassthrough method), 39

short_description() (datalad_next.constraints.compound.EnsureGeneratorFromFileLike method), 39

short_description() (datalad_next.constraints.compound.EnsureIterableOf method), 39

short_description() (datalad_next.constraints.compound.EnsureListOf method), 40

short_description() (datalad_next.constraints.compound.EnsureMapping method), 40

short_description() (datalad_next.constraints.compound.EnsureTupleOf method), 40

short_description() (datalad_next.constraints.compound.WithDescription method), 41

short_description() (datalad_next.constraints.dataset.EnsureDataset method), 44

short_description() (datalad_next.constraints.formats.EnsureJSON method), 41

short_description() (datalad_next.constraints.formats.EnsureURL method), 41

short_description() (datalad_next.constraints.git.EnsureGitRefName method), 44

size (datalad_next.iter_collections.directory.DirectoryItem attribute), 53

size (datalad_next.iter_collections.utils.FileSystemItem attribute), 57

size (datalad_next.types.annexkey.AnnexKey attribute), 61
size
(datalad_next.types.archivist.ArchivistLocator attribute), 61
specialfile
(datalad_next.iter_collections.utils.FileSystemItem attribute), 57
SpecialRemote
(class in datalad_next.annexremotes), 81
SshUrlOperations
(class in datalad_next.url_operations.ssh), 65
stat()
(datalad_next.url_operations.any.AnyUrlOperations method), 63
stat()
(datalad_next.url_operations.file.FileUrlOperations method), 63
stat()
(datalad_next.url_operations.http.HttpUrlOperations method), 65
stat()
(datalad_next.url_operations.ssh.SshUrlOperations method), 65
stat()
(datalad_next.url_operations.UrlOperations method), 68
status_code
(datalad_next.url_operations.UrlOperationsRemoteError property), 70
submodule
(datalad_next.iter_collections.gitworktree.GitTreeItemType attribute), 54
support_githelper_options
(datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote attribute), 76
upload()
(datalad_next.url_operations.ssh.SshUrlOperations method), 66
upload()
(datalad_next.url_operations.UrlOperations method), 68
upload()
(datalad_next.url_operations.UrlOperationsRemoteError property), 70
symlink
(datalad_next.iter_collections.gitworktree.GitTreeItemType attribute), 54
symlink
(datalad_next.iter_collections.utils.FileSystemItemType attribute), 57
TarArchiveOperations
(class in datalad_next.archive_operations.tarfile), 31
tarfile
(datalad_next.archive_operations.tarfile.TarArchiveOperations property), 32
TarfileItem
(class in datalad_next.iter_collections.tarfile), 55
tmp_keyring() (in module datalad_next.tests.fixtures), 60
transfer_retrieve()
(datalad_next.annexremotes.archivist.ArchivistRemote method), 83
transfer_retrieve()
(datalad_next.annexremotes.uncurl.UncurlRemote method), 88
transfer_store()
(datalad_next.annexremotes.archivist.ArchivistRemote method), 83
transfer_store()
(datalad_next.annexremotes.uncurl.UncurlRemote method), 88
tree() (in module datalad.api), 15

valid_property_names_regex
(datalad_next.credman.manager.CredentialManager attribute), 52

Verify_content()
(datalad_next.annexbackends.xdlra.DataladRepoAnnexBackend method), 77

webdav_credential() (in module datalad_next.tests.fixtures), 60
webdav_server() (in module datalad_next.tests.fixtures), 60
WithDescription (class in datalad_next.constraints.compound), 40

X

xdlra_key_locations (datalad_next.gitremotes.datalad_annex.RepoAnnexGitRemote attribute), 76

Z

zip (datalad_next.types.enums.ArchiveType attribute), 62
ZipfileItem (class in datalad_next.iter_collections.zipfile), 56