datalad Documentation

Release 0.19.6+12.g0097136.dirty

DataLad team

Mar 07, 2024

CONTENTS

1 Content 3

Python Module Index 447

Index 449

Welcome to DataLad's **technical documentation**. Information here is targeting software developers and is focused on the Python API and *CLI*, as well as software design, employed technologies, and key features. Comprehensive **user documentation** with information on installation, basic operation, support, and (advanced) use case descriptions is available in the DataLad handbook.

CHAPTER

ONE

CONTENT

1.1 Change log

1.1.1 0.19.6 (2024-02-02)

Enhancements and New Features

- Add the "http_token" authentication mechanism which provides 'Authentication: Token {TOKEN}' header. PR #7551 (by @yarikoptic)

Internal

- Update `pytest_ignore_collect()` for pytest 8.0. PR #7546 (by @jwodder)
- Add manual triggering support/documentation for release workflow. PR #7553 (by @yarikoptic)

1.1.2 0.19.5 (2023-12-28)

Tests

- Fix text to account for a recent change in git-annex dropping sub-second clock precision. As a result we might not report push of git-annex branch since there would be none. PR #7544 (by @yarikoptic)

1.1.3 0.19.4 (2023-12-13)

Bug Fixes

- Update target detection for adjusted mode datasets has been improved. Fixes #7507 via PR #7522 (by @mih)
- Fix typos found by new codespell 2.2.6 and also add checking/fixing "hidden files". PR #7530 (by @yarikoptic)

Documentation

- Improve threaded-runner documentation. Fixes #7498 via PR #7500 (by @christian-monch)

Internal

- add RRID to package metadata. PR #7495 (by @adswa)
- Fix time_diff* and time_remove benchmarks to account for long RFed interfaces. PR #7502 (by @yarikoptic)

Tests

- Cache value of the has_symlink_capability to spare some cycles. PR #7471 (by @yarikoptic)
- RF(TST): use setup_method and teardown_method in TestAddArchiveOptions. PR #7488 (by @yarikoptic)
- Announce test_clone_datasets_root xfail on github osx. PR #7489 (by @yarikoptic)
- Inform asv that there should be no warmup runs for time_remove benchmark. PR #7505 (by @yarikoptic)
- BF(TST): Relax matching of git-annex error message about unsafe drop, which was changed in 10.20231129-18-gfd0b510573. PR #7541 (by @yarikoptic)

1.1.4 0.19.3 (2023-08-10)

Bug Fixes

- Type annotate get_status_dict and note that we can pass Exception or CapturedException which is not subclass. PR #7403 (by @yarikoptic)
- BF: create-sibling-gitlab used to raise a TypeError when attempting a recursive operation in a dataset with uninstalled subdatasets. It now raises an impossible result instead. PR #7430 (by @adswa)
- Pass branch option into recursive call within Install - for the cases whenever install is invoked with URL(s). Fixes #7461 via PR #7463 (by @yarikoptic)
- Allow for reckless=ephemeral clone using relative path for the original location. Fixes #7469 via PR #7472 (by @yarikoptic)

Documentation

- Fix a property name and default costs described in "getting subdatasets" section of `get` documentation. Fixes #7458 via PR #7460 (by @mslw)

Internal

- Copy an adjusted environment only if requested to do so. PR #7399 (by @christian-monch)
- Eliminate uses of `pkg_resources`. Fixes #7435 via PR #7439 (by @jwodder)

Tests

- Disable some S3 tests of their VCR taping where they fail for known issues. PR #7467 (by @yarikoptic)

1.1.5 0.19.2 (2023-07-03)

Bug Fixes

- Remove surrounding quotes in output filenames even for newer version of annex. Fixes #7440 via PR #7443 (by @yarikoptic)

Documentation

- DOC: clarify description of the "install" interface to reflect its convoluted behavior. PR #7445 (by @yarikoptic)

1.1.6 0.19.1 (2023-06-26)

Internal

- Make compatible with upcoming release of git-annex (next after 10.20230407) and pass explicit core.quotepath=false to all git calls. Also added `tools/find-hanged-tests` helper. PR #7372 (by @yarikoptic)

Tests

- Adjust tests for upcoming release of git-annex (next after 10.20230407) and ignore DeprecationWarning for pkg_resources for now. PR #7372 (by @yarikoptic)

1.1.7 0.19.0 (2023-06-14)

Enhancements and New Features

- Address gitlab API special character restrictions. PR #7407 (by @jsheunis)
- BF: The default layout of create-sibling-gitlab is now `collection`. The previous default, `hierarchy` has been removed as it failed in –recursive mode in different edgecases. For single-level datasets, the outcome of `collection` and `hierarchy` is identical. PR #7410 (by @jsheunis and @adswa)

Bug Fixes

- WTF - bring back and extend information on metadata extractors etc, and allow for sections to have subsections and be selected at both levels PR #7309 (by @yarikoptic)
- BF: Run an actual git invocation with interactive commit config. PR #7398 (by @adswa)

Dependencies

- Raise minimal version of tqdm (progress bars) to v.4.32.0 PR #7330 (by @mslw)

Documentation

- DOC: Add a "User messaging" design doc. PR #7310 (by @jsheunis)

Tests

- Remove nose-based testing utils and possibility to test extensions using nose. PR #7261 (by @yarikoptic)

1.1.8 0.18.5 (2023-06-13)

Bug Fixes

- More correct summary reporting for relaxed (no size) –annex. PR #7050 (by @yarikoptic)
- ENH: minor tune up of addurls to be more tolerant and "informative". PR #7388 (by @yarikoptic)
- Ensure that data generated by timeout handlers in the asynchronous runner are accessible via the result generator, even if no other other events occur. PR #7390 (by @christian-monch)
- Do not map (leave as is) trailing / or in github URLs. PR #7418 (by @yarikoptic)

Documentation

- Use `sphinx_autodoc_typehints`. Fixes #7404 via PR #7412 (by @jwodder)

Internal

- Discontinue ConfigManager abuse for Git identity warning. PR #7378 (by @mih) and PR #7392 (by @yarikoptic)

Tests

- Boost python to 3.8 during extensions testing. PR #7413 (by @yarikoptic)
- Skip test_system_ssh_version if no ssh found + split parsing into separate test. PR #7422 (by @yarikoptic)

1.1.9 0.18.4 (2023-05-16)

Bug Fixes

- Provider config files were ignored, when CWD changed between different datasets during runtime. Fixes #7347 via PR #7357 (by @bpoldrack)

Documentation

- Added a workaround for an issue with documentation theme (search function not working on Read the Docs). Fixes #7374 via PR #7385 (by @mslw)

Internal

- Type-annotate `datalad/support/gitrepo.py`. PR #7341 (by @jwodder)

Tests

- Fix failing testing on CI PR #7379 (by @yarikoptic)
  - use sample S3 url DANDI archive,
  - use our copy of old .deb from datasets.datalad.org instead of snapshots.d.o
  - use specific miniconda installer for py 3.7.

1.1.10 0.18.3 (2023-03-25)

Bug Fixes

- Fixed that the `get` command would fail, when subdataset source-candidate-templates where using the `path` property from `.gitmodules`. Also enhance the respective documentation for the `get` command. Fixes #7274 via PR #7280 (by @bpoldrack)
- Improve up-to-dateness of config reports across manager instances. Fixes #7299 via PR #7301 (by @mih)
- BF: GitRepo.merge do not allow merging unrelated unconditionally. PR #7312 (by @yarikoptic)
- Do not render (empty) WTF report on other records. PR #7322 (by @yarikoptic)
- Fixed a bug where changing DataLad's log level could lead to failing git-annex calls. Fixes #7328 via PR #7329 (by @bpoldrack)
- Fix an issue with uninformative error reporting by the datalad special remote. Fixes #7332 via PR #7333 (by @bpoldrack)
- Fix save to not force committing into git if reference dataset is pure git (not git-annex). Fixes #7351 via PR #7355 (by @yarikoptic)

Documentation

- Include a few previously missing commands in html API docs. Fixes #7288 via PR #7289 (by @mslw)

Internal

- Type-annotate almost all of `datalad/utils.py`; add `datalad/typing.py`. PR #7317 (by @jwodder)
- Type-annotate and fix `datalad/support/strings.py`. PR #7318 (by @jwodder)
- Type-annotate `datalad/support/globbedpaths.py`. PR #7327 (by @jwodder)
- Extend type-annotations for `datalad/support/path.py`. PR #7336 (by @jwodder)
- Type-annotate various things in `datalad/runner/`. PR #7337 (by @jwodder)
- Type-annotate some more files in `datalad/support/`. PR #7339 (by @jwodder)

Tests

- Skip or xfail some currently failing or stalling tests. PR #7331 (by @yarikoptic)
- Skip with_sameas_remote when rsync and annex are incompatible. Fixes #7320 via PR #7342 (by @bpoldrack)
- Fix testing assumption - do create pure GitRepo superdataset and test against it. PR #7353 (by @yarikoptic)

1.1.11 0.18.2 (2023-02-27)

Bug Fixes

- Fix `create-sibling` for non-English SSH remotes by providing `LC_ALL=C` for the `ls` call. PR #7265 (by @nobodyinperson)
- Fix EnsureListOf() and EnsureTupleOf() for string inputs. PR #7267 (by @nobodyinperson)
- create-sibling: Use C.UTF-8 locale instead of C on the remote end. PR #7273 (by @nobodyinperson)
- Address compatibility with most recent git-annex where info would exit with non-0. PR #7292 (by @yarikoptic)

Dependencies

- Revert "Revert"Remove chardet version upper limit"". PR #7263 (by @yarikoptic)

Internal

- Codespell more (CHANGELOGs etc) and remove custom CLI options from tox.ini. PR #7271 (by @yarikoptic)

Tests

- Use older python 3.8 in testing nose utils in github-action test-nose. Fixes #7259 via PR #7260 (by @yarikoptic)

1.1.12 0.18.1 (2023-01-16)

Bug Fixes

- Fixes crashes on windows where DataLad was mistaking git-annex 10.20221212 for a not yet released git-annex version and trying to use a new feature. Fixes #7248 via PR #7249 (by @bpoldrack)

Documentation

- DOC: fix EnsureCallable docstring. PR #7245 (by @matrss)

Performance

- Integrate buffer size optimization from datalad-next, leading to significant performance improvement for status and diff. Fixes #7190 via PR #7250 (by @bpoldrack)

1.1.13 0.18.0 (2022-12-31)

Breaking Changes

- Move all old-style metadata commands `aggregate_metadata`, `search`, `metadata` and `extract-metadata`, as well as the `cfg_metadatatypes` procedure and the old metadata extractors into the datalad-deprecated extension. Now recommended way of handling metadata is to install the datalad-metalad extension instead. Fixes #7012 via PR #7014
- Automatic reconfiguration of the ORA special remote when cloning from RIA stores now only applies locally rather than being committed. PR #7235 (by @bpoldrack)

Enhancements and New Features

- A repository description can be specified with a new `--description` option when creating siblings using `create-sibling-[gin|gitea|github|gogs]`. Fixes #6816 via PR #7109 (by @mslw)
- Make validation failure of alternative constraints more informative. Fixes #7092 via PR #7132 (by @bpoldrack)
- Saving removed dataset content was sped-up, and reporting of types of removed content now accurately states `dataset` for added and removed subdatasets, instead of `file`. Moreover, saving previously staged deletions is now also reported. PR #6784 (by @mih)
- `foreach-dataset` command got a new possible value for the –output-streamns|–o-s option 'relpath' to capture and pass-through prefixing with path to subds. Very handy for e.g. running `git grep` command across subdatasets. PR #7071 (by @yarikoptic)
- New config `datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE` allows to add and/or overwrite local configuration for the created sibling by the commands `create-sibling-<gin|gitea|github|gitlab|gogs>`. PR #7213 (by @matrss)
- The `siblings` command does not concern the user with messages about inconsequential failure to annex-enable a remote anymore. PR #7217 (by @bpoldrack)
- ORA special remote now allows to override its configuration locally. PR #7235 (by @bpoldrack)
- Added a 'ria' special remote to provide backwards compatibility with datasets that were set up with the deprecated ria-remote. PR #7235 (by @bpoldrack)

Bug Fixes

- When `create-sibling-ria` was invoked with a sibling name of a pre-existing sibling, a duplicate key in the result record caused a crashed. Fixes #6950 via PR #6952 (by @adswa)

Documentation

- create-sibling-ria's docstring now defines the schema of RIA URLs and clarifies internal layout of a RIA store. PR #6861 (by @adswa)
- Move maintenance team info from issue to CONTRIBUTING. PR #6904 (by @adswa)
- Describe specifications for a DataLad GitHub Action. PR #6931 (by @thewtex)
- Fix capitalization of some service names. PR #6936 (by @aqw)
- Command categories in help text are more consistently named. PR #7027 (by @aqw)
- DOC: Add design document on Tests and CI. PR #7195 (by @adswa)
- CONTRIBUTING.md was extended with up-to-date information on CI logging, changelog and release procedures. PR #7204 (by @yarikoptic)

Internal

- Allow EnsureDataset constraint to handle Path instances. Fixes #7069 via PR #7133 (by @bpoldrack)
- Use `looseversion.LooseVersion` as drop-in replacement for `distutils.version.LooseVersion` Fixes #6307 via PR #6839 (by @effigies)
- Use –pathspec-from-file where possible instead of passing long lists of paths to git/git-annex calls. Fixes #6922 via PR #6932 (by @yarikoptic)
- Make clone_dataset() better patchable ny extensions and less monolithic. PR #7017 (by @mih)
- Remove `simplejson` in favor of using `json`. Fixes #7034 via PR #7035 (by @christian-monch)
- Fix an error in the command group names-test. PR #7044 (by @christian-monch)
- Move eval_results() into interface.base to simplify imports for command implementations. Deprecate use from interface.utils accordingly. Fixes #6694 via PR #7170 (by @adswa)

Performance

- Use regular dicts instead of OrderedDicts for speedier operations. Fixes #6566 via PR #7174 (by @adswa)
- Reimplement `get_submodules_()` without `get_content_info()` for substantial performance boosts especially for large datasets with few subdatasets. Originally proposed in PR #6942 by @mih, fixing #6940. PR #7189 (by @adswa). Complemented with PR #7220 (by @yarikoptic) to avoid `O(N^2)` (instead of `O(N*log(N))` performance in some cases.
- Use –include=* or –anything instead of –copies 0 to speed up get_content_annexinfo. PR #7230 (by @yarikoptic)

Tests

- Re-enable two now-passing core test on Windows CI. PR #7152 (by @adswa)
- Remove the `with_testrepos` decorator and associated tests for it Fixes #6752 via PR #7176 (by @adswa)

1.1.14 0.17.10 (2022-12-14)

Enhancements and New Features

- Enhance concurrent invocation behavior of `ThreadedRunner.run()`. If possible invocations are serialized instead of raising *re-enter* runtime errors. Deadlock situations are detected and runtime errors are raised instead of deadlocking. Fixes #7138 via PR #7201 (by @christian-monch)
- Exceptions bubbling up through CLI are now reported on including their chain of **cause**. Fixes #7163 via PR #7210 (by @bpoldrack)

Bug Fixes

- BF: read RIA config from stdin instead of temporary file. Fixes #6514 via PR #7147 (by @adswa)
- Prevent doomed annex calls on files we already know are untracked. Fixes #7032 via PR #7166 (by @adswa)
- Comply to Posix-like clone URL formats on Windows. Fixes #7180 via PR #7181 (by @adswa)
- Ensure that paths used in the datalad-url field of .gitmodules are posix. Fixes #7182 via PR #7183 (by @adswa)
- Bandaids for export-to-figshare to restore functionality. PR #7188 (by @adswa)
- Fixes hanging threads when `close()` or `del` where called in `BatchedCommand` instances. That could lead to hanging tests if the tests used the `@serve_path_via_http()`-decorator Fixes #6804 via PR #7201 (by @christian-monch)
- Interpret file-URL path components according to the local operating system as described in RFC 8089. With this fix, `datalad.network.RI('file:...').localpath` returns a correct local path on Windows if the RI is constructed with a file-URL. Fixes #7186 via PR #7206 (by @christian-monch)
- Fix a bug when retrieving several files from a RIA store via SSH, when the annex key does not contain size information. Fixes #7214 via PR #7215 (by @mslw)
- Interface-specific (python vs CLI) doc generation for commands and their parameters was broken when brackets were used within the interface markups. Fixes #7225 via PR #7226 (by @bpoldrack)

Documentation

- Fix documentation of `Runner.run()` to not accept strings. Instead, encoding must be ensured by the caller. Fixes #7145 via PR #7155 (by @bpoldrack)

Internal

- Fix import of the `ls` command from datalad-deprecated for benchmarks. Fixes #7149 via PR #7154 (by @bpoldrack)
- Unify definition of parameter choices with `datalad clean`. Fixes #7026 via PR #7161 (by @bpoldrack)

Tests

- Fix test failure with old annex. Fixes #7157 via PR #7159 (by @bpoldrack)
- Re-enable now passing test_path_diff test on Windows. Fixes #3725 via PR #7194 (by @yarikoptic)
- Use Plaintext keyring backend in tests to avoid the need for (interactive) authentication to unlock the keyring during (CI-) test runs. Fixes #6623 via PR #7209 (by @bpoldrack)

1.1.15 0.17.9 (2022-11-07)

Bug Fixes

- Various small fixups ran after looking post-release and trying to build Debian package. PR #7112 (by @yarikoptic)
- BF: Fix add-archive-contents try-finally statement by defining variable earlier. PR #7117 (by @adswa)
- Fix RIA file URL reporting in exception handling. PR #7123 (by @adswa)
- HTTP download treated '429 - too many requests' as an authentication issue and was consequently trying to obtain credentials. Fixes #7129 via PR #7129 (by @bpoldrack)

Dependencies

- Unrestrict pytest and pytest-cov versions. PR #7125 (by @jwodder)
- Remove remaining references to `nose` and the implied requirement for building the documentation Fixes #7100 via PR #7136 (by @bpoldrack)

Internal

- Use datalad/release-action. Fixes #7110. PR #7111 (by @jwodder)
- Fix all logging to use %-interpolation and not .format, sort imports in touched files, add pylint-ing for % formatting in log messages to `tox -e lint`. PR #7118 (by @yarikoptic)

Tests

- Increase the upper time limit after which we assume that a process is stalling. That should reduce false positives from `datalad.support.tests.test_parallel.py::test_stalling`, without impacting the runtime of passing tests. PR #7119 (by @christian-monch)
- XFAIL a check on length of results in test_gracefull_death. PR #7126 (by @yarikoptic)
- Configure Git to allow for "file" protocol in tests. PR #7130 (by @yarikoptic)

1.1.16 0.17.8 (2022-10-24)

Bug Fixes

- Prevent adding duplicate entries to .gitmodules. PR #7088 (by @yarikoptic)
- [BF] Prevent double yielding of impossible get result Fixes #5537. PR #7093 (by @jsheunis)
- Stop rendering the output of internal `subdatset()` call in the results of `run_procedure()`. Fixes #7091 via PR #7094 (by @mslw & @mih)
- Improve handling of `--existing reconfigure` in `create-sibling-ria`: previously, the command would not make the underlying `git init` call for existing local repositories, leading to some configuration updates not being applied. Partially addresses https://github.com/datalad/datalad/issues/6967 via https://github.com/datalad/datalad/pull/7095 (by @mslw)
- Ensure subprocess environments have a valid path in `os.environ['PWD']`, even if a Path-like object was given to the runner on subprocess creation or invocation. Fixes #7040 via PR #7107 (by @christian-monch)
- Improved reporting when using `dry-run` with github-like `create-sibling*` commands (`-gin`, `-gitea`, `-github`, `-gogs`). The result messages will now display names of the repositories which would be created (useful for recursive operations). PR #7103 (by @mslw)

1.1.17 0.17.7 (2022-10-14)

Bug Fixes

- Let `EnsureChoice` report the value is failed validating. PR #7067 (by @mih)
- Avoid writing to stdout/stderr from within datalad sshrun. This could lead to broken pipe errors when cloning via SSH and was superfluous to begin with. Fixes https://github.com/datalad/datalad/issues/6599 via https://github.com/datalad/datalad/pull/7072 (by @bpoldrack)
- BF: lock across threads check/instantiation of Flyweight instances. Fixes #6598 via PR #7075 (by @yarikoptic)

Internal

- Do not use `gen4`-metadata methods in `datalad metadata`-command. PR #7001 (by @christian-monch)
- Revert "Remove chardet version upper limit" (introduced in 0.17.6~11^2) to bring back upper limit <= 5.0.0 on chardet. Otherwise we can get some deprecation warnings from requests PR #7057 (by @yarikoptic)
- Ensure that `BatchedCommandError` is raised if the subprocesses of `BatchedCommand` fails or raises a `CommandError`. PR #7068 (by @christian-monch)
- RF: remove unused code str-ing PurePath. PR #7073 (by @yarikoptic)
- Update GitHub Actions action versions. PR #7082 (by @jwodder)

Tests

- Fix broken test helpers for result record testing that would falsely pass. PR #7002 (by @bpoldrack)

1.1.18 0.17.6 (2022-09-21)

Bug Fixes

- UX: push - provide specific error with details if push failed due to permission issue. PR #7011 (by @yarikoptic)
- Fix datalad –help to not have *Global options* empty with python 3.10 and list options in "options:" section. PR #7028 (by @yarikoptic)
- Let `create` touch the dataset root, if not saving in parent dataset. PR #7036 (by @mih)
- Let `get_status_dict()` use exception message if none is passed. PR #7037 (by @mih)
- Make choices for `status|diff --annex` and `status|diff --untracked` visible. PR #7039 (by @mih)
- push: Assume 0 bytes pushed if git-annex does not provide bytesize. PR #7049 (by @yarikoptic)

Internal

- Use scriv for CHANGELOG generation in release workflow. PR #7009 (by @jwodder)
- Stop using auto. PR #7024 (by @jwodder)

Tests

- Allow for any 2 from first 3 to be consumed in test_gracefull_death. PR #7041 (by @yarikoptic)

1.1.19 0.17.5 (Fri Sep 02 2022)

Bug Fix

- BF: blacklist 23.9.0 of keyring as introduces regression #7003 (@yarikoptic)
- Make the manpages build reproducible via datalad.source.epoch (to be used in Debian packaging) #6997 (@lamby bot@datalad.org @yarikoptic)
- BF: backquote path/drive in Changelog #6997 (@yarikoptic)

Authors: 3

- Chris Lamb (@lamby)
- DataLad Bot (bot@datalad.org)
- Yaroslav Halchenko (@yarikoptic)

1.1.20 0.17.4 (Tue Aug 30 2022)

Bug Fix

- BF: make logic more consistent for files=[] argument (which is False but not None) #6976 (@yarikoptic)
- Run pytests in parallel (-n 2) on appveyor #6987 (@yarikoptic)
- Add workflow for autogenerating changelog snippets #6981 (@jwodder)
- Provide `/dev/null` (`b:\nul` on Windows) instead of empty string as a git-repo to avoid reading local repo configuration #6986 (@yarikoptic)
- RF: call_from_parser - move code into "else" to simplify reading etc #6982 (@yarikoptic)
- BF: if early attempt to parse resulted in error, setup subparsers #6980 (@yarikoptic)
- Run pytests in parallel (-n 2) on Travis #6915 (@yarikoptic)
- Send one character (no newline) to stdout in protocol test to guarantee a single "message" and thus a single custom value #6978 (@christian-monch)

Tests

- TST: test_stalling – wait x10 not just x5 time #6995 (@yarikoptic)

Authors: 3

- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Yaroslav Halchenko (@yarikoptic)

1.1.21 0.17.3 (Tue Aug 23 2022)

Bug Fix

- BF: git_ignore_check do not overload possible value of stdout/err if present #6937 (@yarikoptic)
- DOCfix: fix docstring GeneratorStdOutErrCapture to say that treats both stdout and stderr identically #6930 (@yarikoptic)
- Explain purpose of create-sibling-ria's –post-update-hook #6958 (@mih)
- ENH+BF: get_parent_paths - make / into sep option and consistently use "/" as path separator #6963 (@yarikoptic)
- BF(TEMP): use git-annex from neurodebian -devel to gain fix for bug detected with datalad-crawler #6965 (@yarikoptic)
- BF(TST): make tests use *path* helper for Windows "friendliness" of the tests #6955 (@yarikoptic)
- BF(TST): prevent auto-upgrade of "remote" test sibling, do not use local path for URL #6957 (@yarikoptic)
- Forbid drop operation from symlink'ed annex (e.g. due to being cloned with –reckless=ephemeral) to prevent data-loss #6959 (@mih)

- Acknowledge git-config comment chars #6944 (@mih @yarikoptic)
- Minor tuneups to please updated codespell #6956 (@yarikoptic)
- TST: Add a testcase for #6950 #6957 (@adswa)
- BF+ENH(TST): fix typo in code of wtf filesystems reports #6920 (@yarikoptic)
- DOC: Datalad -> DataLad #6937 (@aqw)
- BF: fix typo which prevented silently to not show details of filesystems #6930 (@yarikoptic)
- BF(TST): allow for a annex repo version to upgrade if running in adjusted branches #6927 (@yarikoptic)
- RF extensions github action to centralize configuration for extensions etc, use pytest for crawler #6914 (@yarikoptic)
- BF: travis - mark our directory as safe to interact with as root #6919 (@yarikoptic)
- BF: do not pretend we know what repo version git-annex would upgrade to #6902 (@yarikoptic)
- BF(TST): do not expect log message for guessing Path to be possibly a URL on windows #6911 (@yarikoptic)
- ENH(TST): Disable coverage reporting on travis while running pytest #6898 (@yarikoptic)
- RF: just rename internal variable from unclear "op" to "io" #6907 (@yarikoptic)
- DX: Demote loglevel of message on url parameters to DEBUG while guessing RI #6891 (@adswa @yarikoptic)
- Fix and expand datalad.runner type annotations #6893 (@christian-monch @yarikoptic)
- Use pytest to test datalad-metalad in test_extensions-workflow #6892 (@christian-monch)
- Let push honor multiple publication dependencies declared via siblings #6869 (@mih @yarikoptic)
- ENH: upgrade versioneer from versioneer-0.20.dev0 to versioneer-0.23.dev0 #6888 (@yarikoptic)
- ENH: introduce typing checking and GitHub workflow #6885 (@yarikoptic)
- RF,ENH(TST): future proof testing of git annex version upgrade + test annex init on all supported versions #6880 (@yarikoptic)
- ENH(TST): test against supported git annex repo version 10 + make it a full sweep over tests #6881 (@yarikoptic)
- BF: RF f-string uses in logger to %-interpolations #6886 (@yarikoptic)
- Merge branch 'bf-sphinx-5.1.0' into maint #6883 (@yarikoptic)
- BF(DOC): workaround for #10701 of sphinx in 5.1.0 #6883 (@yarikoptic)
- Clarify confusing INFO log message from get() on dataset installation #6871 (@mih)
- Protect again failing to load a command interface from an extension #6879 (@mih)
- Support unsetting config via `datalad -c :<name>` #6864 (@mih)
- Fix DOC string typo in the path within AnnexRepo.annexstatus, and replace with proper sphinx reference #6858 (@christian-monch)
- Improved support for saving typechanges #6793 (@mih)

Pushed to `maint`

- BF: Remove duplicate ds key from result record (@adswa)
- DOC: fix capitalization of service names (@aqw)

Tests

- BF(TST,workaround): just xfail failing archives test on NFS #6912 (@yarikoptic)

Authors: 5

- Adina Wagner (@adswa)
- Alex Waite (@aqw)
- Christian Mnch (@christian-monch)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.22 0.17.2 (Sat Jul 16 2022)

Bug Fix

- BF(TST): do proceed to proper test for error being caught for recent git-annex on windows with symlinks #6850 (@yarikoptic)
- Addressing problem testing against python 3.10 on Travis (skip more annex versions) #6842 (@yarikoptic)
- XFAIL test_runner_parametrized_protocol on python3.8 when getting duplicate output #6837 (@yarikoptic)
- BF: Make create's check for procedures work with several again #6841 (@adswa)
- Support older pytests #6836 (@jwodder)

Authors: 3

- Adina Wagner (@adswa)
- John T. Wodder II (@jwodder)
- Yaroslav Halchenko (@yarikoptic)

1.1.23 0.17.1 (Mon Jul 11 2022)

Bug Fix

- DOC: minor fix - consistent DataLad (not Datalad) in docs and CHANGELOG #6830 (@yarikoptic)
- DOC: fixup/harmonize Changelog for 0.17.0 a little #6828 (@yarikoptic)
- BF: use –python-match minor option in new datalad-installer release to match outside version of Python #6827 (@christian-monch @yarikoptic)
- Do not quote paths for ssh >= 9 #6826 (@christian-monch @yarikoptic)
- Suppress DeprecationWarning to allow for distutils to be used #6819 (@yarikoptic)
- RM(TST): remove testing of datalad.test which was removed from 0.17.0 #6822 (@yarikoptic)
- Avoid import of nose-based tests.utils, make skip_if_no_module() and skip_if_no_network() allowed at module level #6817 (@jwodder)
- BF(TST): use higher level asyncio.run instead of asyncio.get_event_loop in test_inside_async #6808 (@yarikoptic)

Authors: 3

- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Yaroslav Halchenko (@yarikoptic)

1.1.24 0.17.0 (Thu Jul 7 2022) – pytest migration

Enhancements and new features

- "log" progress bar now reports about starting a specific action as well. #6756 (by @yarikoptic)
- Documentation and behavior of traceback reporting for log messages via `DATALAD_LOG_TRACEBACK` was improved to yield a more compact report. The documentation for this feature has been clarified. #6746 (by @mih)
- `datalad unlock` gained a progress bar. #6704 (by @adswa)
- When `create-sibling-gitlab` is called on non-existing subdatasets or paths it now returns an impossible result instead of no feedback at all. #6701 (by @adswa)
- `datalad wtf` includes a report on file system types of commonly used paths. #6664 (by @adswa)
- Use next generation metadata code in search, if it is available. #6518 (by @christian-monch)

Deprecations and removals

- Remove unused and untested log helpers `NoProgressLog` and `OnlyProgressLog`. #6747 (by @mih)
- Remove unused `sorted_files()` helper. #6722 (by @adswa)
- Discontinued the value `stdout` for use with the config variable `datalad.log.target` as its use would inevitably break special remote implementations. #6675 (by @bpoldrack)
- `AnnexRepo.add_urls()` is deprecated in favor of `AnnexRepo.add_url_to_file()` or a direct call to `AnnexRepo.call_annex()`. #6667 (by @mih)
- `datalad test` command and supporting functionality (e.g., `datalad.test`) were removed. #6273 (by @jwodder)

Bug Fixes

- `export-archive` does not rely on `normalize_path()` methods anymore and became more robust when called from subdirectories. #6745 (by @adswa)
- Sanitize keys before checking content availability to ensure that the content availability of files with URL- or custom backend keys is correctly determined and marked. #6663 (by @adswa)
- Ensure saving a new subdataset to a superdataset yields a valid `.gitmodules` record regardless of whether and how a path constraint is given to the `save()` call. Fixes #6547 #6790 (by @mih)
- `save` now repairs annex symlinks broken by a `git-mv` operation prior recording a new dataset state. Fixes #4967 #6795 (by @mih)

Documentation

- API documentation for log helpers, like `log_progress()` is now included in the renderer documentation. #6746 (by @mih)
- New design document on progress reporting. #6734 (by @mih)
- Explain downstream consequences of using `--fast` option in `addurls`. #6684 (by @jdkent)

Internal

- Inline code of `create-sibling-ria` has been refactored to an internal helper to check for siblings with particular names across dataset hierarchies in `datalad-next`, and is reintroduced into core to modularize the code base further. #6706 (by @adswa)
- `get_initialized_logger` now lets a given `logtarget` take precedence over `datalad.log.target`. #6675 (by @bpoldrack)
- Many uses of deprecated call options were replaced with the recommended ones. #6273 (by @jwodder)
- Get rid of `asyncio` import by defining few noops methods from `asyncio.protocols.SubprocessProtocol` directly in `WitlessProtocol`. #6648 (by @yarikoptic)
- Consolidate `GitRepo.remove()` and `AnnexRepo.remove()` into a single implementation. #6783 (by @mih) ## Tests
- Discontinue use of `with_testrepos` decorator other than for the deprecation cycle for `nose`. #6690 (by @mih @bpoldrack) See #6144 for full list of changes.
- Remove usage of deprecated `AnnexRepo.add_urls` in tests. #6683 (by @bpoldrack)

- Minimalistic (adapters, no assert changes, etc) migration from `nose` to `pytest`. Support functionality possibly used by extensions and relying on `nose` helpers is left in place to avoid affecting their run time and defer migration of their test setups.. #6273 (by @jwodder)

Authors: 7

- Yaroslav Halchenko (@yarikoptic)
- Michael Hanke (@mih)
- Benjamin Poldrack (@bpoldrack)
- Adina Wagner (@adswa)
- John T. Wodder (@jwodder)
- Christian Mnch (@christian-monch)
- James Kent (@jdkent)

1.1.25 0.16.7 (Wed Jul 06 2022)

Bug Fix

- Fix broken annex symlink after git-mv before saving + fix a race condition in ssh copy test #6809 (@christian-monch @mih @yarikoptic)
- Do not ignore already known status info on submodules #6790 (@mih)
- Fix "common data source" test to use a valid URL (maint-based & extended edition) #6788 (@mih @yarikoptic)
- Upload coverage from extension tests to Codecov #6781 (@jwodder)
- Clean up line end handling in GitRepo #6768 (@christian-monch)
- Do not skip file-URL tests on windows #6772 (@christian-monch)
- Fix test errors caused by updated chardet v5 release #6777 (@christian-monch)
- Preserve final trailing slash in `call_git()` output #6754 (@adswa @yarikoptic @christian-monch)

Pushed to `maint`

- Make sure a subdataset is saved with a complete .gitmodules record (@mih)

Authors: 5

- Adina Wagner (@adswa)
- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.26 0.16.6 (Tue Jun 14 2022)

Bug Fix

- Prevent duplicated result rendering when searching in default datasets #6765 (@christian-monch)
- BF(workaround): skip test_ria_postclonecfg on OSX for now (@yarikoptic)
- BF(workaround to #6759): if saving credential failed, just log error and continue #6762 (@yarikoptic)
- Prevent reentry of a runner instance #6737 (@christian-monch)

Authors: 2

- Christian Mnch (@christian-monch)
- Yaroslav Halchenko (@yarikoptic)

1.1.27 0.16.5 (Wed Jun 08 2022)

Bug Fix

- BF: push to github - remove datalad-push-default-first config only in non-dry run to ensure we push default branch separately in next step #6750 (@yarikoptic)
- In addition to default (system) ssh version, report configured ssh; fix ssh version parsing on Windows #6729 (@yarikoptic)

Authors: 1

- Yaroslav Halchenko (@yarikoptic)

1.1.28 0.16.4 (Thu Jun 02 2022)

Bug Fix

- BF(TST): RO operations - add test directory into git safe.directory #6726 (@yarikoptic)
- DOC: fixup of docstring for skip_ssh #6727 (@yarikoptic)
- DOC: Set language in Sphinx config to en #6727 (@adswa)
- BF: Catch KeyErrors from unavailable WTF infos #6712 (@adswa)
- Add annex.private to ephemeral clones. That would make git-annex not assign shared (in git-annex branch) annex uuid. #6702 (@bpoldrack @adswa)
- BF: require argcomplete version at least 1.12.3 to test/operate correctly #6693 (@yarikoptic)
- Replace Zenodo DOI with JOSS for due credit #6725 (@adswa)

Authors: 3

- Adina Wagner (@adswa)
- Benjamin Poldrack (@bpoldrack)
- Yaroslav Halchenko (@yarikoptic)

1.1.29 0.16.3 (Thu May 12 2022)

Bug Fix

- No change for a PR to trigger release #6692 (@yarikoptic)
- Sanitize keys before checking content availability to ensure correct value for keys with URL or custom backend #6665 (@adswa @yarikoptic)
- Change a key-value pair in drop result record #6625 (@mslw)
- Link docs of datalad-next #6677 (@mih)
- Fix `GitRepo.get_branch_commits_()` to handle branch names conflicts with paths #6661 (@mih)
- OPT: AnnexJsonProtocol - avoid dragging possibly long data around #6660 (@yarikoptic)
- Remove two too prominent create() INFO log message that duplicate DEBUG log and harmonize some other log messages #6638 (@mih @yarikoptic)
- Remove unsupported parameter create_sibling_ria(existing=None) #6637 (@mih)
- Add released plugin to .autorc to annotate PRs on when released #6639 (@yarikoptic)

Authors: 4

- Adina Wagner (@adswa)
- Michael Hanke (@mih)
- Micha Szczepanik (@mslw)
- Yaroslav Halchenko (@yarikoptic)

1.1.30 0.16.2 (Thu Apr 21 2022)

Bug Fix

- Demote (to level 1 from DEBUG) and speed-up API doc logging (parseParameters) #6635 (@mih)
- Factor out actual data transfer in push #6618 (@christian-monch)
- ENH: include version of datalad in tests teardown Versions: report #6628 (@yarikoptic)
- MNT: Require importlib-metadata >=3.6 for Python < 3.10 for entry_points taking kwargs #6631 (@effigies)
- Factor out credential handling of create-sibling-ghlike #6627 (@mih)
- BF: Fix wrong key name of annex' JSON records #6624 (@bpoldrack)

Pushed to `maint`

- Fix typo in changelog (@mih)
- [ci skip] minor typo fix (@yarikoptic)

Authors: 5

- Benjamin Poldrack (@bpoldrack)
- Chris Markiewicz (@effigies)
- Christian Mnch (@christian-monch)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.31 0.16.1 (Fr Apr 8 2022) – April Fools' Release

- Fixes forgotten changelog in docs

1.1.32 0.16.0 (Fr Apr 8 2022) – Spring cleaning!

Enhancements and new features

- A new set of `create-sibling-*` commands reimplements the GitHub-platform support of `create-sibling-github` and adds support to interface three new platforms in a unified fashion: GIN (`create-sibling-gin`), GOGS (`create-sibling-gogs`), and Gitea (`create-sibling-gitea`). All commands rely on personal access tokens only for authentication, allow for specifying one of several stored credentials via a uniform `--credential` parameter, and support a uniform `--dry-run` mode for testing without network. #5949 (by @mih)
- `create-sibling-github` now has supports direct specification of organization repositories via a `[<org>/]reposyntax` #5949 (by @mih)
- `create-sibling-gitlab` gained a `--dry-run` parameter to match the corresponding parameters in `create-sibling-{github,gin,gogs,gitea}` #6013 (by @adswa)
- The `--new-store-ok` parameter of `create-sibling-ria` only creates new RIA stores when explicitly provided #6045 (by @adswa)
- The default performance of `status()` and `diff()` commands is improved by up to 700% removing file-type evaluation as a default operation, and simplifying the type reporting rule #6097 (by @mih)
- `drop()` and `remove()` were reimplemented in full, conceptualized as the antagonist commands to `get()` and `clone()`. A new, harmonized set of parameters (`--what ['filecontent', 'allkeys', 'datasets', 'all']`, `--reckless ['modification', 'availability', 'undead', 'kill']`) simplifies their API. Both commands include additional safeguards. `uninstall` is replaced with a thin shim command around `drop()` #6111 (by @mih)
- `add_archive_content()` was refactored into a dataset method and gained progress bars #6105 (by @adswa)
- The `datalad` and `datalad-archives` special remotes have been reimplemented based on `AnnexRemote` #6165 (by @mih)

- The `result_renderer()` semantics were decomplexified and harmonized. The previous `default` result renderer was renamed to `generic`. #6174 (by @mih)
- `get_status_dict` learned to include exit codes in the case of CommandErrors #5642 (by @yarikoptic)
- `datalad clone` can now pass options to `git-clone`, adding support for cloning specific tags or branches, naming siblings other names than `origin`, and exposing `git clone`'s optimization arguments #6218 (by @kyleam and @mih)
- Inactive BatchedCommands are cleaned up #6206 (by @jwodder)
- `export-archive-ora` learned to filter files exported to 7z archives #6234 (by @mih and @bpinsard)
- `datalad run` learned to glob recursively #6262 (by @AKSoo)
- The ORA remote learned to recover from interrupted uploads #6267 (by @mih)
- A new threaded runner with support for timeouts and generator-based subprocess communication is introduced and used in `BatchedCommand` and `AnnexRepo` #6244 (by @christian-monch)
- A new switch allows to enable librarymode and queries for the effective API in use #6213 (by @mih)
- `run` and `rerun` now support parallel jobs via `--jobs` #6279 (by @AKSoo)
- A new `foreach-dataset` plumbing command allows to run commands on each (sub)dataset, similar to `git submodule foreach` #5517 (by @yarikoptic)
- The `dataset` parameter is not restricted to only locally resolvable file-URLs anymore #6276 (by @christian-monch)
- DataLad's credential system is now able to query `git-credential` by specifying credential type `git` in the respective provider configuration #5796 (by @bpoldrack)
- DataLad now comes with a git credential helper `git-credential-datalad` allowing Git to query DataLad's credential system #5796 (by @bpoldrack and @mih)
- The new runner now allows for multiple threads #6371 (by @christian-monch)
- A new configurationcommand provides an interface to manipulate and query the DataLad configuration. #6306 (by @mih)
  - Unlike the global Python-only datalad.cfg or dataset-specific Dataset.config configuration managers, this command offers a uniform API across the Python and the command line interfaces.
  - This command was previously available in the mihextras extension as x-configuration, and has been merged into the core package in an improved version. #5489 (by @mih)
  - In its default dump mode, the command provides an annotated list of the effective configuration after considering all configuration sources, including hints on additional configuration settings and their supported values.
- The command line interface help-reporting has been sped up by ~20% #6370 #6378 (by @mih)
- `ConfigManager` now supports reading committed dataset configuration in bare repositories. Analog to reading `.datalad/config` from a worktree, `blob:HEAD:.datalad/config` is read (e.g., the config committed in the default branch). The support includes `reload() change detection using the gitsha of this file. The behavior for non-bare repositories is unchanged. #6332 (by @mih)
- The CLI help generation has been sped up, and now also supports the completion of parameter values for a fixed set of choices #6415 (by @mih)
- Individual command implementations can now declare a specific "on-failure" behavior by defining `Interface.on_failure` to be one of the supported modes (stop, continue, ignore). Previously, such a modification was only possible on a per-call basis. #6430 (by @mih)

- The `run` command changed its default "on-failure" behavior from `continue` to `stop`. This change prevents the execution of a command in case a declared input can not be obtained. Previously, only an error result was yielded (and run eventually yielded a non-zero exit code or an `IncompleteResultsException`), but the execution proceeded and potentially saved a dataset modification despite incomplete inputs, in case the command succeeded. This previous default behavior can still be achieved by calling run with the equivalent of `--on-failure continue` #6430 (by @mih)
- The `run command now provides readily executable, API-specific instructions how to save the results of a command execution that failed expectedly #6434 (by @mih)
- `create-sibling --since=^` mode will now be as fast as `push --since=^` to figure out for which subdatasets to create siblings #6436 (by @yarikoptic)
- When file names contain illegal characters or reserved file names that are incompatible with Windows systems a configurable check for `save` (`datalad.save.windows-compat-warning`) will either do nothing (`none`), emit an incompatibility warning (`warning`, default), or cause `save` to error (`error`) #6291 (by @adswa)
- Improve responsiveness of `datalad drop` in datasets with a large annex. #6580 (by @christian-monch)
- `save` code might operate faster on heavy file trees #6581 (by @yarikoptic)
- Removed a per-file overhead cost for ORA when downloading over HTTP #6609 (by @bpoldrack)
- A new module `datalad.support.extensions` offers the utility functions `register_config()` and `has_config()` that allow extension developers to announce additional configuration items to the central configuration management. #6601 (by @mih)
- When operating in a dirty dataset, `export-to-figshare` now yields and impossible result instead of raising a RunTimeError #6543 (by @adswa)
- Loading DataLad extension packages has been sped-up leading to between 2x and 4x faster run times for loading individual extensions and reporting help output across all installed extensions. #6591 (by @mih)
- Introduces the configuration key `datalad.ssh.executable`. This key allows specifying an ssh-client executable that should be used by datalad to establish ssh-connections. The default value is `ssh` unless on a Windows system where `$WINDIR\System32\OpenSSH\ssh.exe` exists. In this case, the value defaults to `$WINDIR\System32\OpenSSH\ssh.exe`. #6553 (by @christian-monch)
- create-sibling should perform much faster in case of `--since` specification since would consider only submodules related to the changes since that point. #6528 (by @yarikoptic)
- A new configuration setting `datalad.ssh.try-use-annex-bundled-git=yes|no` can be used to influence the default remote git-annex bundle sensing for SSH connections. This was previously done unconditionally for any call to `datalad sshrun` (which is also used for any SSH-related Git or git-annex functionality triggered by DataLad-internal processing) and could incur a substantial per-call runtime cost. The new default is to not perform this sensing, because for, e.g., use as GIT_SSH_COMMAND there is no expectation to have a remote git-annex installation, and even with an existing git-annex/Git bundle on the remote, it is not certain that the bundled Git version is to be preferred over any other Git installation in a user's PATH. #6533 (by @mih)
- `run` now yields a result record immediately after executing a command. This allows callers to use the standard `--on-failure switch` to control whether dataset modifications will be saved for a command that exited with an error. #6447 (by @mih)

Deprecations and removals

- The `--pbs-runner` commandline option (deprecated in `0.15.0`) was removed #5981 (by @mih)
- The dependency to PyGithub was dropped #5949 (by @mih)
- `create-sibling-github`'s credential handling was trimmed down to only allow personal access tokens, because GitHub discontinued user/password based authentication #5949 (by @mih)
- `create-sibling-gitlab`'s `--dryrun` parameter is deprecated in favor or `--dry-run` #6013 (by @adswa)
- Internal obsolete `Gitrepo.*_submodule` methods were moved to `datalad-deprecated` #6010 (by @mih)
- `datalad/support/versions.py` is unused in DataLad core and removed #6115 (by @yarikoptic)
- Support for the undocumented `datalad.api.result-renderer` config setting has been dropped #6174 (by @mih)
- Undocumented use of `result_renderer=None` is replaced with `result_renderer='disabled'` #6174 (by @mih)
- `remove`'s `--recursive` argument has been deprecated #6257 (by @mih)
- The use of the internal helper `get_repo_instance()` is discontinued and deprecated #6268 (by @mih)
- Support for Python 3.6 has been dropped (#6286 (by @christian-monch) and #6364 (by @yarikoptic))
- All but one Singularity recipe flavor have been removed due to their limited value with the end of life of Singularity Hub #6303 (by @mih)
- All code in module datalad.cmdline was (re)moved, only datalad.cmdline.helpers.get_repo_instanceis kept for a deprecation period (by @mih)
- `datalad.interface.common_opts.eval_default` has been deprecated. All (command-specific) defaults for common interface parameters can be read from `Interface` class attributes (#6391 (by @mih)
- Remove unused and untested `datalad.interface.utils` helpers `cls2cmdlinename` and `path_is_under` #6392 (by @mih)
- An unused code path for result rendering was removed from the CLI `main()` #6394 (by @mih)
- `create-sibling` will require now `"^"` instead of an empty string for since option #6436 (by @yarikoptic)
- `run` no longer raises a `CommandError` exception for failed commands, but yields an `error` result that includes a superset of the information provided by the exception. This change impacts command line usage insofar as the exit code of the underlying command is no longer relayed as the exit code of the `run` command call – although `run` continues to exit with a non-zero exit code in case of an error. For Python API users, the nature of the raised exception changes from `CommandError` to `IncompleteResultsError`, and the exception handling is now configurable using the standard `on_failure` command argument. The original `CommandError` exception remains available via the `exception` property of the newly introduced result record for the command execution, and this result record is available via `IncompleteResultsError.failed`, if such an exception is raised. #6447 (by @mih)
- Custom cast helpers were removed from datalad core and migrated to a standalone repository https://github.com/datalad/screencaster #6516 (by @adswa)
- The `bundled` parameter of `get_connection_hash()` is now ignored and will be removed with a future release. #6532 (by @mih)
- `BaseDownloader.fetch()` is logging download attempts on DEBUG (previously INFO) level to avoid polluting output of higher-level commands. #6564 (by @mih)

Bug Fixes

- `create-sibling-gitlab` erroneously overwrote existing sibling configurations. A safeguard will now prevent overwriting and exit with an error result #6015 (by @adswa)
- `create-sibling-gogs` now relays HTTP500 errors, such as "no space left on device" #6019 (by @mih)
- `annotate_paths()` is removed from the last parts of code base that still contained it #6128 (by @mih)
- `add_archive_content()` doesn't crash with `--key` and `--use-current-dir` anymore #6105 (by @adswa)
- `run-procedure` now returns an error result when a non-existent procedure name is specified #6143 (by @mslw)
- A fix for a silent failure of `download-url --archive` when extracting the archive #6172 (by @adswa)
- Uninitialized AnnexRepos can now be dropped #6183 (by @mih)
- Instead of raising an error, the formatters tests are skipped when the `formatters` module is not found #6212 (by @adswa)
- `create-sibling-gin` does not disable git-annex availability on Gin remotes anymore #6230 (by @mih)
- The ORA special remote messaging is fixed to not break the special remote protocol anymore and to better relay messages from exceptions to communicate underlying causes #6242 (by @mih)
- A `keyring.delete()` call was fixed to not call an uninitialized private attribute anymore #6253 (by @bpoldrack)
- An erroneous placement of result keyword arguments into a `format()` method instead of `get_status_dict()` of `create-sibling-ria` has been fixed #6256 (by @adswa)
- `status`, `run-procedure`, and `metadata` are no longer swallowing result-related messages in renderers #6280 (by @mih)
- `uninstall` now recommends the new `--reckless` parameter instead of the deprecated `--nocheck` parameter when reporting hints #6277 (by @adswa)
- `download-url` learned to handle Pathobjects #6317 (by @adswa)
- Restore default result rendering behavior broken by Key interface documentation #6394 (by @mih)
- Fix a broken check for file presence in the `ConfigManager` that could have caused a crash in rare cases when a config file is removed during the process runtime #6332 (by @mih) `- `ConfigManager.get_from_source()` now accesses the correct information when using the documented `source='local'`, avoiding a crash #6332 (by @mih)
- `run` no longer let's the internal call to `save` render its results unconditionally, but the parameterization f run determines the effective rendering format. #6421 (by @mih)
- Remove an unnecessary and misleading warning from the runner #6425 (by @christian-monch)
- A number of commands stopped to double-report results #6446 (by @adswa)
- `create-sibling-ria` no longer creates an `annex/objects` directory in-store, when called with `--no-storage-sibling`. #6495 (by @bpoldrack)
- Improve error message when an invalid URL is given to `clone`. #6500 (by @mih)
- DataLad declares a minimum version dependency to `keyring >= 20.0` to ensure that token-based authentication can be used. #6515 (by @adswa)
- ORA special remote tries to obtain permissions when dropping a key from a RIA store rather than just failing. Thus having the same permissions in the store's object trees as one directly managed by git-annex would have, works just fine now. #6493 (by @bpoldrack)

- require_dataset() now uniformly raises NoDatasetFound when no dataset was found. Implementations that catch the previously documented InsufficientArgumentsError or the actually raised ValueError will continue to work, because NoDatasetFound is derived from both types. #6521 (by @mih)
- Keyboard-interactive authentication is now possibly with non-multiplexed SSH connections (i.e., when no connection sharing is possible, due to lack of socket support, for example on Windows). Previously, it was disabled forcefully by DataLad for no valid reason. #6537 (by @mih)
- Remove duplicate exception type in reporting of top-level CLI exception handler. #6563 (by @mih)
- Fixes DataLad's parsing of git-annex' reporting on unknown paths depending on its version and the value of the annex.skipunknown config. #6550 (by @bpoldrack)
- Fix ORA special remote not properly reporting on HTTP failures. #6535 (by @bpoldrack)
- ORA special remote didn't show per-file progress bars when downloading over HTTP #6609 (by @bpoldrack)
- save now can commit the change where file becomes a directory with a staged for commit file. #6581 (by @yarikoptic)
- create-sibling will no longer create siblings for not yet saved new subdatasets, and will now create subdatasets nested in the subdatasets which did not yet have those siblings. #6603 (by @yarikoptic)

Documentation

- A new design document sheds light on result records #6167 (by @mih)
- The disabled result renderer mode is documented #6174 (by @mih)
- A new design document sheds light on the datalad and datalad-archives special remotes #6181 (by @mih)
- A new design document sheds light on BatchedCommand and BatchedAnnex #6203 (by @christian-monch)
- A new design document sheds light on standard parameters #6214 (by @adswa)
- The DataLad project adopted the Contributor Covenant COC v2.1 #6236 (by @adswa)
- Docstrings learned to include Sphinx' "version added" and "deprecated" directives #6249 (by @mih)
- A design document sheds light on basic docstring handling and formatting #6249 (by @mih)
- A new design document sheds light on position versus keyword parameter usage #6261 (by @yarikoptic)
- create-sibling-gin's examples have been improved to suggest push as an additional step to ensure proper configuration #6289 (by @mslw)
- A new document describes the credential system from a user's perspective #5796 (by @bpoldrack)
- Enhance the design document on DataLad's credential system #5796 (by @bpoldrack)
- The documentation of the configuration command now details all locations DataLad is reading configuration items from, and their respective rules of precedence #6306 (by @mih)
- API docs for datalad.interface.base are now included in the documentation #6378 (by @mih)
- A new design document is provided that describes the basics of the command line interface implementation #6382 (by @mih)
- The `datalad.interface.base.Interface class, the basis of all DataLad command implementations, has been extensively documented to provide an overview of basic principles and customization possibilities #6391 (by @mih)
- --since=^ mode of operation of create-sibling is documented now #6436 (by @yarikoptic)

Internal

- The internal `status()` helper was equipped with docstrings and promotes "breadth-first" reporting with a new parameter `reporting_order` #6006 (by @mih)
- `AnnexRepo.get_file_annexinfo()` is introduced for more convenient queries for single files and replaces a now deprecated `AnnexRepo.get_file_key()` to receive information with fewer calls to Git #6104 (by @mih)
- A new `get_paths_by_ds()` helper exposes `status`' path normalization and sorting #6110 (by @mih)
- `status` is optimized with a cache for dataset roots #6137 (by @yarikoptic)
- The internal `get_func_args_doc()` helper with Python 2 is removed from DataLad core #6175 (by @yarikoptic)
- Further restructuring of the source tree to better reflect the internal dependency structure of the code: `AddArchiveContent` is moved from `datalad/interface` to `datalad/local` (#6188 (by @mih)), `Clean` is moved from `datalad/interface` to `datalad/local` (#6191 (by @mih)), `Unlock` is moved from `datalad/interface` to `datalad/local` (#6192 (by @mih)), `DownloadURL` is moved from `datalad/interface` to `datalad/local` (#6217 (by @mih)), `Rerun` is moved from `datalad/interface` to `datalad/local` (#6220 (by @mih)), `RunProcedure` is moved from `datalad/interface` to `datalad/local` (#6222 (by @mih)). The interface command list is restructured and resorted #6223 (by @mih)
- `wrapt` is replaced with functools' `wraps` #6190 (by @yariktopic)
- The unmaintained `appdirs` library has been replaced with `platformdirs` #6198 (by @adswa)
- Modelines mismatching the code style in source files were fixed #6263 (by @AKSoo)
- `datalad/__init__.py` has been cleaned up #6271 (by @mih)
- `GitRepo.call_git_items` is implemented with a generator-based runner #6278 (by @christian-monch)
- Separate positional from keyword arguments in the Python API to match CLI with * #6176 (by @yarikoptic), #6304 (by @christian-monch)
- `GitRepo.bare` does not require the ConfigManager anymore #6323 (by @mih)
- `_get_dot_git()` was reimplemented to be more efficient and consistent, by testing for common scenarios first and introducing a consistently applied `resolved` flag for result path reporting #6325 (by @mih)
- All data files under `datalad` are now included when installing DataLad #6336 (by @jwodder)
- Add internal method for non-interactive provider/credential storing #5796 (by @bpoldrack)
- Allow credential classes to have a context set, consisting of a URL they are to be used with and a dataset DataLad is operating on, allowing to consider "local" and "dataset" config locations #5796 (by @bpoldrack)
- The Interface method `get_refds_path()` was deprecated #6387 (by @adswa)
- `datalad.interface.base.Interface` is now an abstract class #6391 (by @mih)
- Simplified the decision making for result rendering, and reduced code complexity #6394 (by @mih)
- Reduce code duplication in `datalad.support.json_py` #6398 (by @mih)
- Use public `ArgumentParser.parse_known_args` instead of protected `_parse_known_args` #6414 (by @yarikoptic)
- `add-archive-content` does not rely on the deprecated `tempfile.mktemp` anymore, but uses the more secure `tempfile.mkdtemp` #6428 (by @adswa)
- AnnexRepo's internal `annexstatus` is deprecated. In its place, a new test helper assists the few tests that rely on it #6413 (by @adswa)
- `config` has been refactored from `where[="dataset"]` to `scope[="branch"]` #5969 (by @yarikoptic)

- Common command arguments are now uniformly and exhaustively passed to result renderers and filters for decision making. Previously, the presence of a particular argument depended on the respective API and circumstances of a command call. #6440 (by @mih)
- Entrypoint processing for extensions and metadata extractors has been consolidated on a uniform helper that is about twice as fast as the previous implementations. #6591 (by @mih)

Tests

- A range of Windows tests pass and were enabled #6136 (by @adswa)
- Invalid escape sequences in some tests were fixed #6147 (by @mih)
- A cross-platform compatible HTTP-serving test environment is introduced #6153 (by @mih)
- A new helper exposes `serve_path_via_http` to the command line to deploy an ad-hoc instance of the HTTP server used for internal testing, with SSL and auth, if desired. #6169 (by @mih)
- Windows tests were redistributed across worker runs to harmonize runtime #6200 (by @adswa)
- `Batchedcommand` gained a basic test #6203 (by @christian-monch)
- The use of `with_testrepo` is discontinued in all core tests #6224 (by @mih)
- The new `git-annex.filter.annex.process` configuration is enabled by default on Windows to speed up the test suite #6245 (by @mih)
- If the available Git version supports it, the test suite now uses `GIT_CONFIG_GLOBAL` to configure a fake home directory instead of overwriting `HOME` on OSX (#6251 (by @bpoldrack)) and `HOME` and `USERPROFILE` on Windows #6260 (by @adswa)
- Windows test timeouts of runners were addressed #6311 (by @christian-monch)
- A handful of Windows tests were fixed (#6352 (by @yarikoptic)) or disabled (#6353 (by @yarikoptic))
- `download-url`'s test under `http_proxy` are skipped when a session can't be established #6361 (by @yarikoptic)
- A test for `datalad clean` was fixed to be invoked within a dataset #6359 (by @yarikoptic)
- The new datalad.cli.tests have an improved module coverage of 80% #6378 (by @mih)
- The `test_source_candidate_subdataset` has been marked as `@slow` #6429 (by @yarikoptic)
- Dedicated `CLI` benchmarks exist now #6381 (by @mih)
- Enable code coverage report for subprocesses #6546 (by @adswa)
- Skip a test on annex>=10.20220127 due to a bug in annex. See https://git-annex.branchable.com/bugs/Change_to_annex.largefiles_leaves_repo_modified/

Infra

- A new issue template using GitHub forms prestructures bug reports #6048 (by @Remi-Gau)
- DataLad and its dependency stack were packaged for Gentoo Linux #6088 (by @TheChymera)
- The readthedocs configuration is modernized to version 2 #6207 (by @adswa)
- The Windows CI setup now runs on Appveyor's Visual Studio 2022 configuration #6228 (by @adswa)
- The `readthedocs-theme` and `Sphinx` versions were pinned to re-enable rendering of bullet points in the documentation #6346 (by @adswa)

- The PR template was updated with a CHANGELOG template. Future PRs should use it to include a summary for the CHANGELOG #6396 (by @mih)

Authors: 11

- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)
- Adina Wagner (@adswa)
- Remi Gau (@Remi-Gau)
- Horea Christian (@TheChymera)
- Micha Szczepanik (@mslw)
- Christian Mnch (@christian-monch)
- John T. Wodder (@jwodder)
- Benjamin Poldrack (@bpoldrack)
- Sin Kim (@AKSoo)
- Basile Pinsard (@bpinsard)

1.1.33 0.15.6 (Sun Feb 27 2022)

Bug Fix

- BF: do not use BaseDownloader instance wide InterProcessLock - resolves stalling or errors during parallel installs #6507 (@yarikoptic)
- release workflow: add -vv to auto invocation (@yarikoptic)
- Fix version incorrectly incremented by release process in CHANGELOGs #6459 (@yarikoptic)
- BF(TST): add another condition to skip under http_proxy set #6459 (@yarikoptic)

Authors: 1

- Yaroslav Halchenko (@yarikoptic)

1.1.34 0.15.5 (Wed Feb 09 2022)

Enhancement

- BF: When download-url gets Pathobject as path convert it to a string #6364 (@adswa)

Bug Fix

- Fix AnnexRepo.whereis key=True mode operation, and add batch mode support #6379 (@yarikoptic)
- DOC: run - adjust description for -i/-o to mention that it could be a directory #6416 (@yarikoptic)
- BF: ORA over HTTP tried to check archive #6355 (@bpoldrack @yarikoptic)
- BF: condition access to isatty to have stream eval to True #6360 (@yarikoptic)
- BF: python 3.10 compatibility fixes #6363 (@yarikoptic)
- Remove two(!) copies of a test #6374 (@mih)
- Warn just once about incomplete git config #6343 (@yarikoptic)
- Make version detection robust to GIT_DIR specification #6341 (@effigies @mih)
- BF(Q&D): do not crash - issue warning - if template fails to format #6319 (@yarikoptic)

Authors: 5

- Adina Wagner (@adswa)
- Benjamin Poldrack (@bpoldrack)
- Chris Markiewicz (@effigies)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.35 0.15.4 (Thu Dec 16 2021)

Bug Fix

- BF: autorc - replace incorrect releaseTypes with "none" #6320 (@yarikoptic)
- Minor enhancement to CONTRIBUTING.md #6309 (@bpoldrack)
- UX: If a clean repo is dirty after a failed run, give clean-up hints #6112 (@adswa)
- Stop using distutils #6113 (@jwodder)
- BF: RIARemote - set UI backend to annex to make it interactive #6287 (@yarikoptic @bpoldrack)
- Fix invalid escape sequences #6293 (@jwodder)
- CI: Update environment for windows CI builds #6292 (@bpoldrack)
- bump the python version used for mac os tests #6288 (@christian-monch @bpoldrack)
- ENH(UX): log a hint to use ulimit command in case of "Too long" exception #6173 (@yarikoptic)
- Report correct HTTP URL for RIA store content #6091 (@mih)
- BF: Don't overwrite subdataset source candidates #6168 (@bpoldrack)
- Bump sphinx requirement to bypass readthedocs defaults #6189 (@mih)
- infra: Provide custom prefix to auto-related labels #6151 (@adswa)
- Remove all usage of exc_str() #6142 (@mih)

- BF: obtain information about annex special remotes also from annex journal #6135 (@yarikoptic @mih)
- BF: clone tried to save new subdataset despite failing to clone #6140 (@bpoldrack)

Tests

- RF+BF: use skip_if_no_module helper instead of try/except for libxmp and boto #6148 (@yarikoptic)
- git://github.com -> https://github.com #6134 (@mih)

Authors: 6

- Adina Wagner (@adswa)
- Benjamin Poldrack (@bpoldrack)
- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.36 0.15.3 (Sat Oct 30 2021)

Bug Fix

- BF: Don't make create-sibling recursive by default #6116 (@adswa)
- BF: Add dashes to 'force' option in non-empty directory error message #6078 (@DisasterMo)
- DOC: Add supported URL types to download-url's docstring #6098 (@adswa)
- BF: Retain git-annex error messages & don't show them if operation successful #6070 (@DisasterMo)
- Remove uses of `__full_version__` and `datalad.version` #6073 (@jwodder)
- BF: ORA shouldn't crash while handling a failure #6063 (@bpoldrack)
- DOC: Refine –reckless docstring on usage and wording #6043 (@adswa)
- BF: archives upon strip - use rmtree which retries etc instead of rmdir #6064 (@yarikoptic)
- BF: do not leave test in a tmp dir destined for removal #6059 (@yarikoptic)
- Next wave of exc_str() removals #6022 (@mih)

Pushed to `maint`

- CI: Enable new codecov uploader in Appveyor CI (@adswa)

Internal

- UX: Log clone-candidate number and URLs #6092 (@adswa)
- UX/ENH: Disable reporting, and don't do superfluous internal subdatasets calls #6094 (@adswa)
- Update codecov action to v2 #6072 (@jwodder)

Documentation

- Design document on URL substitution feature #6065 (@mih)

Tests

- BF(TST): remove reuse of the same tape across unrelated tests #6127 (@yarikoptic)
- Fail Travis tests on deprecation warnings #6074 (@jwodder)
- Ux get result handling broken #6052 (@christian-monch)
- enable metalad tests again #6060 (@christian-monch)

Authors: 7

- Adina Wagner (@adswa)
- Benjamin Poldrack (@bpoldrack)
- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Michael Burgardt (@DisasterMo)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.37 0.15.2 (Wed Oct 06 2021)

Bug Fix

- BF: Don't suppress datalad subdatasets output #6035 (@DisasterMo @mih)
- Honor datalad.runtime.use-patool if set regardless of OS (was Windows only) #6033 (@mih)
- Discontinue usage of deprecated (public) helper #6032 (@mih)
- BF: ProgressHandler - close the other handler if was specified #6020 (@yarikoptic)
- UX: Report GitLab weburl of freshly created projects in the result #6017 (@adswa)

- Ensure there's a blank line between the class `__doc__` and "Parameters" in `build_doc` docstrings #6004 (@jwodder)
- Large code-reorganization of everything runner-related #6008 (@mih)
- Discontinue exc_str() in all modern parts of the code base #6007 (@mih)

Tests

- TST: Add test to ensure functionality with subdatasets starting with a hyphen (-) #6042 (@DisasterMo)
- BF(TST): filter away warning from coverage from analysis of stderr of –help #6028 (@yarikoptic)
- BF: disable outdated SSL root certificate breaking chain on older/buggy clients #6027 (@yarikoptic)
- BF: start global test_http_server only if not running already #6023 (@yarikoptic)

Authors: 5

- Adina Wagner (@adswa)
- John T. Wodder II (@jwodder)
- Michael Burgardt (@DisasterMo)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.38 0.15.1 (Fri Sep 24 2021)

Bug Fix

- BF: downloader - fail to download even on non-crippled FS if symlink exists #5991 (@yarikoptic)
- ENH: import datalad.api to bind extensions methods for discovery of dataset methods #5999 (@yarikoptic)
- Restructure cmdline API presentation #5988 (@mih)
- Close file descriptors after process exit #5983 (@mih)

Pushed to `maint`

- Discontinue testing of hirni extension (@mih)

Internal

- Add debugging information to release step #5980 (@jwodder)

Documentation

- Coarse description of the credential subsystem's functionality #5998 (@mih)

Tests

- BF(TST): use sys.executable, mark test_ria_basics.test_url_keys as requiring network #5986 (@yarikoptic)

Authors: 3

- John T. Wodder II (@jwodder)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.39 0.15.0 (Tue Sep 14 2021) – We miss you Kyle!

Enhancements and new features

- Command execution is now performed by a new `Runner` implementation that is no longer based on the `asyncio` framework, which was found to exhibit fragile performance in interaction with other `asyncio`-using code, such as Jupyter notebooks. The new implementation is based on threads. It also supports the specification of "protocols" that were introduced with the switch to the `asyncio` implementation in 0.14.0. (#5667)
- `clone` now supports arbitrary URL transformations based on regular expressions. One or more transformation steps can be defined via `datalad.clone.url-substitute.<label>` configuration settings. The feature can be (and is now) used to support convenience mappings, such as `https://osf.io/q8xnk/` (displayed in a browser window) to `osf://q8xnk` (clonable via the `datalad-osf` extension. (#5749)
- Homogenize SSH use and configurability between DataLad and git-annex, by instructing git-annex to use DataLad's `sshrun` for SSH calls (instead of SSH directly). (#5389)
- The ORA special remote has received several new features:
  - It now support a `push-url` setting as an alternative to `url` for write access. An analog parameter was also added to `create-sibling-ria`. (#5420, #5428)
  - Access of RIA stores now performs homogeneous availability checks, regardless of access protocol. Before, broken HTTP-based access due to misspecified URLs could have gone unnoticed. (#5459, #5672)
  - Error reporting was introduce to inform about undesirable conditions in remote RIA stores. (#5683)
- `create-sibling-ria` now supports `--alias` for the specification of a convenience dataset alias name in a RIA store. (#5592)
- Analog to `git commit`, `save` now features an `--amend` mode to support incremental updates of a dataset state. (#5430)
- `run` now supports a dry-run mode that can be used to inspect the result of parameter expansion on the effective command to ease the composition of more complicated command lines. (#5539)
- `run` now supports a `--assume-ready` switch to avoid the (possibly expensive) preparation of inputs and outputs with large datasets that have already been readied through other means. (#5431)

- update now features --how and --how-subds parameters to configure how an update shall be performed. Supported modes are fetch (unchanged default), and merge (previously also possible via --merge), but also new strategies like reset or checkout. (#5534)
- update has a new --follow=parentds-lazy mode that only performs a fetch operation in subdatasets when the desired commit is not yet present. During recursive updates involving many subdatasets this can substantially speed up performance. (#5474)
- DataLad's command line API can now report the version for individual commands via datalad <cmd> --version. The output has been homogenized to <providing package> <version>. (#5543)
- create-sibling now logs information on an auto-generated sibling name, in the case that no --name/-s was provided. (#5550)
- create-sibling-github has been updated to emit result records like any standard DataLad command. Previously it was implemented as a "plugin", which did not support all standard API parameters. (#5551)
- copy-file now also works with content-less files in datasets on crippled filesystems (adjusted mode), when a recent enough git-annex (8.20210428 or later) is available. (#5630)
- addurls can now be instructed how to behave in the event of file name collision via a new parameter --on-collision. (#5675)
- addurls reporting now informs which particular subdatasets were created. (#5689)
- Credentials can now be provided or overwritten via all means supported by ConfigManager. Importantly, datalad.credential.<name>.<field> configuration settings and analog specification via environment variables are now supported (rather than custom environment variables only). Previous specification methods are still supported too. (#5680)
- A new datalad.credentials.force-ask configuration flag can now be used to force re-entry of already known credentials. This simplifies credential updates without having to use an approach native to individual credential stores. (#5777)
- Suppression of rendering repeated similar results is now configurable via the configuration switches datalad.ui.suppress-similar-results (bool), and datalad.ui.suppress-similar-results-threshold (int). (#5681)
- The performance of status and similar functionality when determining local file availability has been improved. (#5692)
- push now renders a result summary on completion. (#5696)
- A dedicated info log message indicates when dataset repositories are subjected to an annex version upgrade. (#5698)
- Error reporting improvements:
  - The NoDatasetFound exception now provides information for which purpose a dataset is required. (#5708)
  - Wording of the MissingExternalDependeny error was rephrased to account for cases of non-functional installations. (#5803)
  - push reports when a --to parameter specification was (likely) forgotten. (#5726)
  - Detailed information is now given when DataLad fails to obtain a lock for credential entry in a timely fashion. Previously only a generic debug log message was emitted. (#5884)
  - Clarified error message when create-sibling-gitlab was called without --project. (#5907)
- add-readme now provides a README template with more information on the nature and use of DataLad datasets. A README file is no longer annex'ed by default, but can be using the new --annex switch. ([#5723][], [#5725][])

- `clean` now supports a `--dry-run` mode to inform about cleanable content. (#5738)
- A new configuration setting `datalad.locations.locks` can be used to control the placement of lock files. (#5740)
- `wtf` now also reports branch names and states. (#5804)
- `AnnexRepo.whereis()` now supports batch mode. (#5533)

Deprecations and removals

- The minimum supported git-annex version is now 8.20200309. (#5512)
- ORA special remote configuration items `ssh-host`, and `base-path` are deprecated. They are completely replaced by `ria+<protocol>://` URL specifications. (#5425)
- The deprecated `no_annex` parameter of `create()` was removed from the Python API. (#5441)
- The unused `GitRepo.pull()` method has been removed. (#5558)
- Residual support for "plugins" (a mechanism used before DataLad supported extensions) was removed. This includes the configuration switches `datalad.locations.{system,user}-plugins`. (#5554, #5564)
- Several features and comments have been moved to the `datalad-deprecated` package. This package must now be installed to be able to use keep using this functionality.
  - The `publish` command. Use `push` instead. (#5837)
  - The `ls` command. (#5569)
  - The web UI that is deployable via `datalad create-sibling --ui`. (#5555)
  - The "automagic IO" feature. (#5577)
- `AnnexRepo.copy_to()` has been deprecated. The `push` command should be used instead. (#5560)
- `AnnexRepo.sync()` has been deprecated. `AnnexRepo.call_annex(['sync', ...])` should be used instead. (#5461)
- All `GitRepo.*_submodule()` methods have been deprecated and will be removed in a future release. (#5559)
- `create-sibling-github`'s `--dryrun` switch was deprecated, use `--dry-run` instead. (#5551)
- The `datalad --pbs-runner` option has been deprecated, use `condor_run` (or similar) instead. (#5956)

Fixes

- Prevent invalid declaration of a publication dependencies for 'origin' on any auto-detected ORA special remotes, when cloing from a RIA store. An ORA remote is now checked whether it actually points to the RIA store the clone was made from. (#5415)
- The ORA special remote implementation has received several fixes:
  - It can now handle HTTP redirects. (#5792)
  - Prevents failure when URL-type annex keys contain the '/' character. (#5823)
  - Properly support the specification of usernames, passwords and ports in `ria+<protocol>://` URLs. (#5902)
- It is now possible to specifically select the default (or generic) result renderer via `datalad -f default` and with that override a `tailored` result renderer that may be preconfigured for a particular command. (#5476)

- Starting with 0.14.0, original URLs given to `clone` were recorded in a subdataset record. This was initially done in a second commit, leading to inflation of commits and slowdown in superdatasets with many subdatasets. Such subdataset record annotation is now collapsed into a single commits. (#5480)
- `run` now longer removes leading empty directories as part of the output preparation. This was surprising behavior for commands that do not ensure on their own that output directories exist. (#5492)
- A potentially existing `message` property is no longer removed when using the `json` or `json_pp` result renderer to avoid undesired withholding of relevant information. (#5536)
- `subdatasets` now reports `state=present`, rather than `state=clean`, for installed subdatasets to complement `state=absent` reports for uninstalled dataset. (#5655)
- `create-sibling-ria` now executes commands with a consistent environment setup that matches all other command execution in other DataLad commands. (#5682)
- `save` no longer saves unspecified subdatasets when called with an explicit path (list). The fix required a behavior change of `GitRepo.get_content_info()` in its interpretation of `None` vs. `[]` path argument values that now aligns the behavior of `GitRepo.diff|status()` with their respective documentation. (#5693)
- `get` now prefers the location of a subdatasets that is recorded in a superdataset's `.gitmodules` record. Previously, DataLad tried to obtain a subdataset from an assumed checkout of the superdataset's origin. This new default order is (re-)configurable via the `datalad.get.subdataset-source-candidate-<priority-label>` configuration mechanism. (#5760)
- `create-sibling-gitlab` no longer skips the root dataset when `.` is given as a path. (#5789)
- `siblings` now rejects a value given to `--as-common-datasrc` that clashes with the respective Git remote. (#5805)
- The usage synopsis reported by `siblings` now lists all supported actions. (#5913)
- `siblings` now renders non-ok results to avoid silent failure. (#5915)
- `.gitattribute` file manipulations no longer leave the file without a trailing newline. (#5847)
- Prevent crash when trying to delete a non-existing keyring credential field. (#5892)
- git-annex is no longer called with an unconditional `annex.retry=3` configuration. Instead, this parameterization is now limited to `annex get` and `annex copy` calls. (#5904)

Tests

- `file://` URLs are no longer the predominant test case for `AnnexRepo` functionality. A built-in HTTP server now used in most cases. (#5332)

1.1.40 0.14.8 (Sun Sep 12 2021)

Bug Fix

- BF: add-archive-content on .xz and other non-.gz stream compressed files #5930 (@yarikoptic)
- BF(UX): do not keep logging ERROR possibly present in progress records #5936 (@yarikoptic)
- Annotate datalad_core as not needing actual data – just uses annex whereis #5971 (@yarikoptic)
- BF: limit CMD_MAX_ARG if obnoxious value is encountered. #5945 (@yarikoptic)

- Download session/credentials locking – inform user if locking is "failing" to be obtained, fail upon ~5min timeout #5884 (@yarikoptic)
- Render siblings()'s non-ok results with the default renderer #5915 (@mih)
- BF: do not crash, just skip whenever trying to delete non existing field in the underlying keyring #5892 (@yarikoptic)
- Fix argument-spec for `siblings` and improve usage synopsis #5913 (@mih)
- Clarify error message re unspecified gitlab project #5907 (@mih)
- Support username, password and port specification in RIA URLs #5902 (@mih)
- BF: take path from SSHRI, test URLs not only on Windows #5881 (@yarikoptic)
- ENH(UX): warn user if keyring returned a "null" keyring #5875 (@yarikoptic)
- ENH(UX): state original purpose in NoDatasetFound exception + detail it for get #5708 (@yarikoptic)

Pushed to `maint`

- Merge branch 'bf-http-headers-agent' into maint (@yarikoptic)
- RF(BF?)+DOC: provide User-Agent to entire session headers + use those if provided (@yarikoptic)

Internal

- Pass `--no-changelog` to `auto shipit` if changelog already has entry #5952 (@jwodder)
- Add isort config to match current convention + run isort via pre-commit (if configured) #5923 (@jwodder)
- .travis.yml: use python -m {nose,coverage} invocations, and always show combined report #5888 (@yarikoptic)
- Add project URLs into the package metadata for convenience links on Pypi #5866 (@adswa @yarikoptic)

Tests

- BF: do use OBSCURE_FILENAME instead of hardcoded unicode #5944 (@yarikoptic)
- BF(TST): Skip testing for having PID listed if no psutil #5920 (@yarikoptic)
- BF(TST): Boost version of git-annex to 8.20201129 to test an error message #5894 (@yarikoptic)

Authors: 4

- Adina Wagner (@adswa)
- John T. Wodder II (@jwodder)
- Michael Hanke (@mih)
- Yaroslav Halchenko (@yarikoptic)

1.1.41 0.14.7 (Tue Aug 03 2021)

Bug Fix

- UX: When two or more clone URL templates are found, error out more gracefully #5839 (@adswa)
- BF: http_auth - follow redirect (just 1) to re-authenticate after initial attempt #5852 (@yarikoptic)
- addurls Formatter - provide value repr in exception #5850 (@yarikoptic)
- ENH: allow for "patch" level semver for "master" branch #5839 (@yarikoptic)
- BF: Report info from annex JSON error message in CommandError #5809 (@mih)
- RF(TST): do not test for no EASY and pkg_resources in shims #5817 (@yarikoptic)
- http downloaders: Provide custom informative User-Agent, do not claim to be "Authenticated access" #5802 (@yarikoptic)
- ENH(UX,DX): inform user with a warning if version is 0+unknown #5787 (@yarikoptic)
- shell-completion: add argcomplete to 'misc' extra_depends, log an ERROR if argcomplete fails to import #5781 (@yarikoptic)
- ENH (UX): add python-gitlab dependency #5776 (s.heunis@fz-juelich.de)

Internal

- BF: Fix reported paths in ORA remote #5821 (@adswa)
- BF: import importlib.metadata not importlib_metadata whenever available #5818 (@yarikoptic)

Tests

- TST: set –allow-unrelated-histories in the mk_push_target setup for Windows #5855 (@adswa)
- Tests: Allow for version to contain + as a separator and provide more information for version related comparisons #5786 (@yarikoptic)

Authors: 4

- Adina Wagner (@adswa)
- Michael Hanke (@mih)
- Stephan Heunis (@jsheunis)
- Yaroslav Halchenko (@yarikoptic)

1.1.42 0.14.6 (Sun Jun 27 2021)

Internal

- BF: update changelog conversion from .md to .rst (for sphinx) #5757 (@yarikoptic @jwodder)

Authors: 2

- John T. Wodder II (@jwodder)
- Yaroslav Halchenko (@yarikoptic)

1.1.43 0.14.5 (Mon Jun 21 2021)

Bug Fix

- BF(TST): parallel - take longer for producer to produce #5747 (@yarikoptic)
- add –on-failure default value and document it #5690 (@christian-monch @yarikoptic)
- ENH: harmonize "purpose" statements to imperative form #5733 (@yarikoptic)
- ENH(TST): populate heavy tree with 100 unique keys (not just 1) among 10,000 #5734 (@yarikoptic)
- BF: do not use .acquired - just get state from acquire() #5718 (@yarikoptic)
- BF: account for annex now "scanning for annexed" instead of "unlocked" files #5705 (@yarikoptic)
- interface: Don't repeat custom summary for non-generator results #5688 (@kyleam)
- RF: just pip install datalad-installer #5676 (@yarikoptic)
- DOC: addurls.extract: Drop mention of removed 'stream' parameter #5690 (@kyleam)
- Merge pull request #5674 from kyleam/test-addurls-copy-fix #5674 (@kyleam)
- Merge pull request #5663 from kyleam/status-ds-equal-path #5663 (@kyleam)
- Merge pull request #5671 from kyleam/update-fetch-fail #5671 (@kyleam)
- BF: update: Honor –on-failure if fetch fails #5671 (@kyleam)
- RF: update: Avoid fetch's deprecated kwargs #5671 (@kyleam)
- CLN: update: Drop an unused import #5671 (@kyleam)
- Merge pull request #5664 from kyleam/addurls-better-url-parts-error #5664 (@kyleam)
- Merge pull request #5661 from kyleam/sphinx-fix-plugin-refs #5661 (@kyleam)
- BF: status: Provide special treatment of "this dataset" path #5663 (@kyleam)
- BF: addurls: Provide better placeholder error for special keys #5664 (@kyleam)
- RF: addurls: Simply construction of placeholder exception message #5664 (@kyleam)
- RF: addurls._get_placeholder_exception: Rename a parameter #5664 (@kyleam)
- RF: status: Avoid repeated Dataset.path access #5663 (@kyleam)
- DOC: Reference plugins via datalad.api #5661 (@kyleam)
- download-url: Set up datalad special remote if needed #5648 (@kyleam @yarikoptic)

Pushed to `maint`

- MNT: Post-release dance (@kyleam)

Internal

- Switch to versioneer and auto #5669 (@jwodder @yarikoptic)
- MNT: setup.py: Temporarily avoid Sphinx 4 #5649 (@kyleam)

Tests

- BF(TST): skip testing for showing "Scanning for ..." since not shown if too quick #5727 (@yarikoptic)
- Revert "TST: test_partial_unlocked: Document and avoid recent git-annex failure" #5651 (@kyleam)

Authors: 4

- Christian Mnch (@christian-monch)
- John T. Wodder II (@jwodder)
- Kyle Meyer (@kyleam)
- Yaroslav Halchenko (@yarikoptic)

1.1.44 0.14.4 (May 10, 2021) – .

Fixes

- Following an internal call to `git-clone`, clone assumed that the remote name was "origin", but this may not be the case if `clone.defaultRemoteName` is configured (available as of Git 2.30). (#5572)
- Several test fixes, including updates for changes in git-annex. (#5612) (#5632) (#5639)

1.1.45 0.14.3 (April 28, 2021) – .

Fixes

- For outputs that include a glob, run didn't re-glob after executing the command, which is necessary to catch changes if `--explicit` or `--expand={outputs,both}` is specified. (#5594)
- run now gives an error result rather than a warning when an input glob doesn't match. (#5594)
- The procedure for creating a RIA store checks for an existing ria-layout-version file and makes sure its version matches the desired version. This check wasn't done correctly for SSH hosts. (#5607)
- A helper for transforming git-annex JSON records into DataLad results didn't account for the unusual case where the git-annex record doesn't have a "file" key. (#5580)
- The test suite required updates for recent changes in PyGithub and git-annex. (#5603) (#5609)

Enhancements and new features

- The DataLad source repository has long had a tools/cmdline-completion helper. This functionality is now exposed as a command, `datalad shell-completion`. (#5544)

1.1.46 0.14.2 (April 14, 2021) – .

Fixes

- push now works bottom-up, pushing submodules first so that hooks on the remote can aggregate updated subdataset information. (#5416)
- run-procedure didn't ensure that the configuration of subdatasets was reloaded. (#5552)

1.1.47 0.14.1 (April 01, 2021) – .

Fixes

- The recent default branch changes on GitHub's side can lead to "git-annex" being selected over "master" as the default branch on GitHub when setting up a sibling with create-sibling-github. To work around this, the current branch is now pushed first. (#5010)
- The logic for reading in a JSON line from git-annex failed if the response exceeded the buffer size (256 KB on *nix systems).
- Calling unlock with a path of "." from within an untracked subdataset incorrectly aborted, complaining that the "dataset containing given paths is not underneath the reference dataset". (#5458)
- clone didn't account for the possibility of multiple accessible ORA remotes or the fact that none of them may be associated with the RIA store being cloned. (#5488)
- create-sibling-ria didn't call `git update-server-info` after setting up the remote repository and, as a result, the repository couldn't be fetched until something else (e.g., a push) triggered a call to `git update-server-info`. (#5531)
- The parser for git-config output didn't properly handle multi-line values and got thrown off by unexpected and unrelated lines. (#5509)
- The 0.14 release introduced regressions in the handling of progress bars for git-annex actions, including collapsing progress bars for concurrent operations. (#5421) (#5438)
- save failed if the user configured Git's `diff.ignoreSubmodules` to a non-default value. (#5453)
- A interprocess lock is now used to prevent a race between checking for an SSH socket's existence and creating it. (#5466)
- If a Python procedure script is executable, run-procedure invokes it directly rather than passing it to `sys.executable`. The non-executable Python procedures that ship with DataLad now include shebangs so that invoking them has a chance of working on file systems that present all files as executable. (#5436)
- DataLad's wrapper around `argparse` failed if an underscore was used in a positional argument. (#5525)

Enhancements and new features

- DataLad's method for mapping environment variables to configuration options (e.g., `DATALAD_FOO_X__Y` to `datalad.foo.x-y`) doesn't work if the subsection name ("FOO") has an underscore. This limitation can be sidestepped with the new `DATALAD_CONFIG_OVERRIDES_JSON` environment variable, which can be set to a JSON record of configuration values. (#5505)

1.1.48 0.14.0 (February 02, 2021) – .

Major refactoring and deprecations

- Git versions below v2.19.1 are no longer supported. (#4650)
- The minimum git-annex version is still 7.20190503, but, if you're on Windows (or use adjusted branches in general), please upgrade to at least 8.20200330 but ideally 8.20210127 to get subdataset-related fixes. (#4292) (#5290)
- The minimum supported version of Python is now 3.6. (#4879)
- publish is now deprecated in favor of push. It will be removed in the 0.15.0 release at the earliest.
- A new command runner was added in v0.13. Functionality related to the old runner has now been removed: `Runner`, `GitRunner`, and `run_gitcommand_on_file_list_chunks` from the `datalad.cmd` module along with the `datalad.tests.protocolremote`, `datalad.cmd.protocol`, and `datalad.cmd.protocol.prefix` configuration options. (#5229)
- The `--no-storage-sibling` switch of `create-sibling-ria` is deprecated in favor of `--storage-sibling=off` and will be removed in a later release. (#5090)
- The `get_git_dir` static method of `GitRepo` is deprecated and will be removed in a later release. Use the `dot_git` attribute of an instance instead. (#4597)
- The `ProcessAnnexProgressIndicators` helper from `datalad.support.annexrepo` has been removed. (#5259)
- The `save` argument of install, a noop since v0.6.0, has been dropped. (#5278)
- The `get_URLS` method of `AnnexCustomRemote` is deprecated and will be removed in a later release. (#4955)
- `ConfigManager.get` now returns a single value rather than a tuple when there are multiple values for the same key, as very few callers correctly accounted for the possibility of a tuple return value. Callers can restore the old behavior by passing `get_all=True`. (#4924)
- In 0.12.0, all of the `assure_*` functions in `datalad.utils` were renamed as `ensure_*`, keeping the old names around as compatibility aliases. The `assure_*` variants are now marked as deprecated and will be removed in a later release. (#4908)
- The `datalad.interface.run` module, which was deprecated in 0.12.0 and kept as a compatibility shim for `datalad.core.local.run`, has been removed. (#4583)
- The `saver` argument of `datalad.core.local.run.run_command`, marked as obsolete in 0.12.0, has been removed. (#4583)
- The `dataset_only` argument of the `ConfigManager` class was deprecated in 0.12 and has now been removed. (#4828)
- The `linux_distribution_name`, `linux_distribution_release`, and `on_debian_wheezy` attributes in `datalad.utils` are no longer set at import time and will be removed in a later release. Use `datalad.utils.get_linux_distribution` instead. (#4696)

- `datalad.distribution.clone`, which was marked as obsolete in v0.12 in favor of `datalad.core.distributed.clone`, has been removed. (#4904)
- `datalad.support.annexrepo.N_AUTO_JOBS`, announced as deprecated in v0.12.6, has been removed. (#4904)
- The `compat` parameter of `GitRepo.get_submodules`, added in v0.12 as a temporary compatibility layer, has been removed. (#4904)
- The long-deprecated (and non-functional) `url` parameter of `GitRepo.__init__` has been removed. (#5342)

Fixes

- Cloning onto a system that enters adjusted branches by default (as Windows does) did not properly record the clone URL. (#5128)
- The RIA-specific handling after calling clone was correctly triggered by `ria+http` URLs but not `ria+https` URLs. (#4977)
- If the registered commit wasn't found when cloning a subdataset, the failed attempt was left around. (#5391)
- The remote calls to `cp` and `chmod` in create-sibling were not portable and failed on macOS. (#5108)
- A more reliable check is now done to decide if configuration files need to be reloaded. (#5276)
- The internal command runner's handling of the event loop has been improved to play nicer with outside applications and scripts that use asyncio. (#5350) (#5367)

Enhancements and new features

- The subdataset handling for adjusted branches, which is particularly important on Windows where git-annex enters an adjusted branch by default, has been improved. A core piece of the new approach is registering the commit of the primary branch, not its checked out adjusted branch, in the superdataset. Note: This means that `git status` will always consider a subdataset on an adjusted branch as dirty while `datalad status` will look more closely and see if the tip of the primary branch matches the registered commit. (#5241)
- The performance of the subdatasets command has been improved, with substantial speedups for recursive processing of many subdatasets. (#4868) (#5076)
- Adding new subdatasets via save has been sped up. (#4793)
- get, save, and addurls gained support for parallel operations that can be enabled via the `--jobs` command-line option or the new `datalad.runtime.max-jobs` configuration option. (#5022)
- addurls
  - learned how to read data from standard input. (#4669)
  - now supports tab-separated input. (#4845)
  - now lets Python callers pass in a list of records rather than a file name. (#5285)
  - gained a `--drop-after` switch that signals to drop a file's content after downloading and adding it to the annex. (#5081)
  - is now able to construct a tree of files from known checksums without downloading content via its new `--key` option. (#5184)
  - records the URL file in the commit message as provided by the caller rather than using the resolved absolute path. (#5091)
  - is now speedier. (#4867) (#5022)

- create-sibling-github learned how to create private repositories (thanks to Nolan Nichols). (#4769)
- create-sibling-ria gained a `--storage-sibling` option. When `--storage-sibling=only` is specified, the storage sibling is created without an accompanying Git sibling. This enables using hosts without Git installed for storage. (#5090)
- The download machinery (and thus the `datalad` special remote) gained support for a new scheme, `shub://`, which follows the same format used by `singularity run` and friends. In contrast to the short-lived URLs obtained by querying Singularity Hub directly, `shub://` URLs are suitable for registering with git-annex. (#4816)
- A provider is now included for https://registry-1.docker.io URLs. This is useful for storing an image's blobs in a dataset and registering the URLs with git-annex. (#5129)
- The `add-readme` command now links to the DataLad handbook rather than http://docs.datalad.org. (#4991)
- New option `datalad.locations.extra-procedures` specifies an additional location that should be searched for procedures. (#5156)
- The class for handling configuration values, `ConfigManager`, now takes a lock before writes to allow for multiple processes to modify the configuration of a dataset. (#4829)
- clone now records the original, unresolved URL for a subdataset under `submodule.<name>.datalad-url` in the parent's .gitmodules, enabling later get calls to use the original URL. This is particularly useful for `ria+` URLs. (#5346)
- Installing a subdataset now uses custom handling rather than calling `git submodule update --init`. This avoids some locking issues when running get in parallel and enables more accurate source URLs to be recorded. (#4853)
- `GitRepo.get_content_info`, a helper that gets triggered by many commands, got faster by tweaking its `git ls-files` call. (#5067)
- wtf now includes credentials-related information (e.g. active backends) in the its output. (#4982)
- The `call_git*` methods of `GitRepo` now have a `read_only` parameter. Callers can set this to `True` to promise that the provided command does not write to the repository, bypassing the cost of some checks and locking. (#5070)
- New `call_annex*` methods in the `AnnexRepo` class provide an interface for running git-annex commands similar to that of the `GitRepo.call_git*` methods. (#5163)
- It's now possible to register a custom metadata indexer that is discovered by search and used to generate an index. (#4963)
- The `ConfigManager` methods `get`, `getbool`, `getfloat`, and `getint` now return a single value (with same precedence as `git config --get`) when there are multiple values for the same key (in the non-committed git configuration, if the key is present there, or in the dataset configuration). For `get`, the old behavior can be restored by specifying `get_all=True`. (#4924)
- Command-line scripts are now defined via the `entry_points` argument of `setuptools.setup` instead of the `scripts` argument. (#4695)
- Interactive use of `--help` on the command-line now invokes a pager on more systems and installation setups. (#5344)
- The `datalad` special remote now tries to eliminate some unnecessary interactions with git-annex by being smarter about how it queries for URLs associated with a key. (#4955)
- The `GitRepo` class now does a better job of handling bare repositories, a step towards bare repositories support in DataLad. (#4911)
- More internal work to move the code base over to the new command runner. (#4699) (#4855) (#4900) (#4996) (#5002) (#5141) (#5142) (#5229)

1.1.49 0.13.7 (January 04, 2021) – .

Fixes

- Cloning from a RIA store on the local file system initialized annex in the Git sibling of the RIA source, which is problematic because all annex-related functionality should go through the storage sibling. clone now sets `remote.origin.annex-ignore` to `true` after cloning from RIA stores to prevent this. (#5255)
- create-sibling invoked `cp` in a way that was not compatible with macOS. (#5269)
- Due to a bug in older Git versions (before 2.25), calling status with a file under .git/ (e.g., `datalad status .git/config`) incorrectly reported the file as untracked. A workaround has been added. (#5258)
- Update tests for compatibility with latest git-annex. (#5254)

Enhancements and new features

- copy-file now aborts if .git/ is in the target directory, adding to its existing .git/ safety checks. (#5258)

1.1.50 0.13.6 (December 14, 2020) – .

Fixes

- An assortment of fixes for Windows compatibility. (#5113) (#5119) (#5125) (#5127) (#5136) (#5201) (#5200) (#5214)
- Adding a subdataset on a system that defaults to using an adjusted branch (i.e. doesn't support symlinks) didn't properly set up the submodule URL if the source dataset was not in an adjusted state. (#5127)
- push failed to push to a remote that did not have an `annex-uuid` value in the local `.git/config`. (#5148)
- The default renderer has been improved to avoid a spurious leading space, which led to the displayed path being incorrect in some cases. (#5121)
- siblings showed an uninformative error message when asked to configure an unknown remote. (#5146)
- drop confusingly relayed a suggestion from `git annex drop` to use `--force`, an option that does not exist in `datalad drop`. (#5194)
- create-sibling-github no longer offers user/password authentication because it is no longer supported by GitHub. (#5218)
- The internal command runner's handling of the event loop has been tweaked to hopefully fix issues with running DataLad from IPython. (#5106)
- SSH cleanup wasn't reliably triggered by the ORA special remote on failure, leading to a stall with a particular version of git-annex, 8.20201103. (This is also resolved on git-annex's end as of 8.20201127.) (#5151)

Enhancements and new features

- The credential helper no longer asks the user to repeat tokens or AWS keys. (#5219)
- The new option `datalad.locations.sockets` controls where DataLad stores SSH sockets, allowing users to more easily work around file system and path length restrictions. (#5238)

1.1.51 0.13.5 (October 30, 2020) – .

Fixes

- SSH connection handling has been reworked to fix cloning on Windows. A new configuration option, `datalad.ssh.multiplex-connections`, defaults to false on Windows. (#5042)
- The ORA special remote and post-clone RIA configuration now provide authentication via DataLad's credential mechanism and better handling of HTTP status codes. (#5025) (#5026)
- By default, if a git executable is present in the same location as git-annex, DataLad modifies `PATH` when running git and git-annex so that the bundled git is used. This logic has been tightened to avoid unnecessarily adjusting the path, reducing the cases where the adjustment interferes with the local environment, such as special remotes in a virtual environment being masked by the system-wide variants. (#5035)
- git-annex is now consistently invoked as "git annex" rather than "git-annex" to work around failures on Windows. (#5001)
- push called `git annex sync ...` on plain git repositories. (#5051)
- save in genernal doesn't support registering multiple levels of untracked subdatasets, but it can now properly register nested subdatasets when all of the subdataset paths are passed explicitly (e.g., `datalad save -d. sub-a sub-a/sub-b`). (#5049)
- When called with `--sidecar` and `--explicit`, run didn't save the sidecar. (#5017)
- A couple of spots didn't properly quote format fields when combining substrings into a format string. (#4957)
- The default credentials configured for `indi-s3` prevented anonymous access. (#5045)

Enhancements and new features

- Messages about suppressed similar results are now rate limited to improve performance when there are many similar results coming through quickly. (#5060)
- create-sibling-github can now be told to replace an existing sibling by passing `--existing=replace`. (#5008)
- Progress bars now react to changes in the terminal's width (requires tqdm 2.1 or later). (#5057)

1.1.52 0.13.4 (October 6, 2020) – .

Fixes

- Ephemeral clones mishandled bare repositories. (#4899)
- The post-clone logic for configuring RIA stores didn't consider `https://` URLs. (#4977)
- DataLad custom remotes didn't escape newlines in messages sent to git-annex. (#4926)
- The datalad-archives special remote incorrectly treated file names as percent-encoded. (#4953)
- The result handler didn't properly escape "%" when constructing its message template. (#4953)

- In v0.13.0, the tailored rendering for specific subtypes of external command failures (e.g., "out of space" or "remote not available") was unintentionally switched to the default rendering. (#4966)
- Various fixes and updates for the NDA authenticator. (#4824)
- The helper for getting a versioned S3 URL did not support anonymous access or buckets with "." in their name. (#4985)
- Several issues with the handling of S3 credentials and token expiration have been addressed. (#4927) (#4931) (#4952)

Enhancements and new features

- A warning is now given if the detected Git is below v2.13.0 to let users that run into problems know that their Git version is likely the culprit. (#4866)
- A fix to push in v0.13.2 introduced a regression that surfaces when `push.default` is configured to "matching" and prevents the git-annex branch from being pushed. Note that, as part of the fix, the current branch is now always pushed even when it wouldn't be based on the configured refspec or `push.default` value. (#4896)
- publish
  - now allows spelling the empty string value of `--since=` as `^` for consistency with push. (#4683)
  - compares a revision given to `--since=` with `HEAD` rather than the working tree to speed up the operation. (#4448)
- rerun
  - emits more INFO-level log messages. (#4764)
  - provides better handling of adjusted branches and aborts with a clear error for cases that are not supported. (#5328)
- The archives are handled with p7zip, if available, since DataLad v0.12.0. This implementation now supports .tgz and .tbz2 archives. (#4877)

1.1.53 0.13.3 (August 28, 2020) – .

Fixes

- Work around a Python bug that led to our asyncio-based command runner intermittently failing to capture the output of commands that exit very quickly. (#4835)
- push displayed an overestimate of the transfer size when multiple files pointed to the same key. (#4821)
- When download-url calls `git annex addurl`, it catches and reports any failures rather than crashing. A change in v0.12.0 broke this handling in a particular case. (#4817)

Enhancements and new features

- The wrapper functions returned by decorators are now given more meaningful names to hopefully make tracebacks easier to digest. (#4834)

1.1.54 0.13.2 (August 10, 2020) – .

Deprecations

- The `allow_quick` parameter of `AnnexRepo.file_has_content` and `AnnexRepo.is_under_annex` is now ignored and will be removed in a later release. This parameter was only relevant for git-annex versions before 7.20190912. (#4736)

Fixes

- Updates for compatibility with recent git and git-annex releases. (#4746) (#4760) (#4684)
- push didn't sync the git-annex branch when `--data=nothing` was specified. (#4786)
- The `datalad.clone.reckless` configuration wasn't stored in non-annex datasets, preventing the values from being inherited by annex subdatasets. (#4749)
- Running the post-update hook installed by `create-sibling --ui` could overwrite web log files from previous runs in the unlikely event that the hook was executed multiple times in the same second. (#4745)
- clone inspected git's standard error in a way that could cause an attribute error. (#4775)
- When cloning a repository whose `HEAD` points to a branch without commits, clone tries to find a more useful branch to check out. It unwisely considered adjusted branches. (#4792)
- Since v0.12.0, `SSHManager.close` hasn't closed connections when the `ctrl_path` argument was explicitly given. (#4757)
- When working in a dataset in which `git annex init` had not yet been called, the `file_has_content` and `is_under_annex` methods of `AnnexRepo` incorrectly took the "allow quick" code path on file systems that did not support it (#4736)

Enhancements

- create now assigns version 4 (random) UUIDs instead of version 1 UUIDs that encode the time and hardware address. (#4790)
- The documentation for create now does a better job of describing the interaction between `--dataset` and `PATH`. (#4763)
- The `format_commit` and `get_hexsha` methods of `GitRepo` have been sped up. (#4807) (#4806)
- A better error message is now shown when the `^` or `^.` shortcuts for `--dataset` do not resolve to a dataset. (#4759)
- A more helpful error message is now shown if a caller tries to download an `ftp://` link but does not have `request_ftp` installed. (#4788)
- clone now tries harder to get up-to-date availability information after auto-enabling `type=git` special remotes. (#2897)

1.1.55 0.13.1 (July 17, 2020) – .

Fixes

- Cloning a subdataset should inherit the parent's `datalad.clone.reckless` value, but that did not happen when cloning via `datalad get` rather than `datalad install` or `datalad clone`. (#4657)
- The default result renderer crashed when the result did not have a `path` key. (#4666) (#4673)
- `datalad push` didn't show information about `git push` errors when the output was not in the format that it expected. (#4674)
- `datalad push` silently accepted an empty string for `--since` even though it is an invalid value. (#4682)
- Our JavaScript testing setup on Travis grew stale and has now been updated. (Thanks to Xiao Gui.) (#4687)
- The new class for running Git commands (added in v0.13.0) ignored any changes to the process environment that occurred after instantiation. (#4703)

Enhancements and new features

- `datalad push` now avoids unnecessary `git push` dry runs and pushes all refspecs with a single `git push` call rather than invoking `git push` for each one. (#4692) (#4675)
- The readability of SSH error messages has been improved. (#4729)
- `datalad.support.annexrepo` avoids calling `datalad.utils.get_linux_distribution` at import time and caches the result once it is called because, as of Python 3.8, the function uses `distro` underneath, adding noticeable overhead. (#4696)

  Third-party code should be updated to use `get_linux_distribution` directly in the unlikely event that the code relied on the import-time call to `get_linux_distribution` setting the `linux_distribution_name`, `linux_distribution_release`, or `on_debian_wheezy` attributes in `datalad.utils.

1.1.56 0.13.0 (June 23, 2020) – .

A handful of new commands, including `copy-file`, `push`, and `create-sibling-ria`, along with various fixes and enhancements

Major refactoring and deprecations

- The `no_annex` parameter of create, which is exposed in the Python API but not the command line, is deprecated and will be removed in a later release. Use the new `annex` argument instead, flipping the value. Command-line callers that use `--no-annex` are unaffected. (#4321)
- `datalad add`, which was deprecated in 0.12.0, has been removed. (#4158) (#4319)
- The following `GitRepo` and `AnnexRepo` methods have been removed: `get_changed_files`, `get_missing_files`, and `get_deleted_files`. (#4169) (#4158)
- The `get_branch_commits` method of `GitRepo` and `AnnexRepo` has been renamed to `get_branch_commits_`. (#3834)
- The custom `commit` method of `AnnexRepo` has been removed, and `AnnexRepo.commit` now resolves to the parent method, `GitRepo.commit`. (#4168)
- GitPython's `git.repo.base.Repo` class is no longer available via the `.repo` attribute of `GitRepo` and `AnnexRepo`. (#4172)

- `AnnexRepo.get_corresponding_branch` now returns `None` rather than the current branch name when a managed branch is not checked out. (#4274)
- The special UUID for git-annex web remotes is now available as `datalad.consts.WEB_SPECIAL_REMOTE_UUID`. It remains accessible as `AnnexRepo.WEB_UUID` for compatibility, but new code should use `consts.WEB_SPECIAL_REMOTE_UUID` (#4460).

Fixes

- Widespread improvements in functionality and test coverage on Windows and crippled file systems in general. (#4057) (#4245) (#4268) (#4276) (#4291) (#4296) (#4301) (#4303) (#4304) (#4305) (#4306)
- `AnnexRepo.get_size_from_key` incorrectly handled file chunks. (#4081)
- create-sibling would too readily clobber existing paths when called with `--existing=replace`. It now gets confirmation from the user before doing so if running interactively and unconditionally aborts when running non-interactively. (#4147)
- update (#4159)
  - queried the incorrect branch configuration when updating non-annex repositories.
  - didn't account for the fact that the local repository can be configured as the upstream "remote" for a branch.
- When the caller included `--bare` as a `git init` option, create crashed creating the bare repository, which is currently unsupported, rather than aborting with an informative error message. (#4065)
- The logic for automatically propagating the 'origin' remote when cloning a local source could unintentionally trigger a fetch of a non-local remote. (#4196)
- All remaining `get_submodules()` call sites that relied on the temporary compatibility layer added in v0.12.0 have been updated. (#4348)
- The custom result summary renderer for get, which was visible with `--output-format=tailored`, displayed incorrect and confusing information in some cases. The custom renderer has been removed entirely. (#4471)
- The documentation for the Python interface of a command listed an incorrect default when the command overrode the value of command parameters such as `result_renderer`. (#4480)

Enhancements and new features

- The default result renderer learned to elide a chain of results after seeing ten consecutive results that it considers similar, which improves the display of actions that have many results (e.g., saving hundreds of files). (#4337)
- The default result renderer, in addition to "tailored" result renderer, now triggers the custom summary renderer, if any. (#4338)
- The new command create-sibling-ria provides support for creating a sibling in a RIA store. (#4124)
- DataLad ships with a new special remote, git-annex-remote-ora, for interacting with RIA stores and a new command export-archive-ora for exporting an archive from a local annex object store. (#4260) (#4203)
- The new command push provides an alternative interface to publish for pushing a dataset hierarchy to a sibling. (#4206) (#4581) (#4617) (#4620)
- The new command copy-file copies files and associated availability information from one dataset to another. (#4430)
- The command examples have been expanded and improved. (#4091) (#4314) (#4464)
- The tooling for linking to the DataLad Handbook from DataLad's documentation has been improved. (#4046)

- The `--reckless` parameter of clone and install learned two new modes:
  - "ephemeral", where the .git/annex/ of the cloned repository is symlinked to the local source repository's. (#4099)
  - "shared-{group|all|...}" that can be used to set up datasets for collaborative write access. (#4324)
- clone
  - learned to handle dataset aliases in RIA stores when given a URL of the form `ria+<protocol>://<storelocation>#~<aliasname>`. (#4459)
  - now checks `datalad.get.subdataset-source-candidate-NAME` to see if `NAME` starts with three digits, which is taken as a "cost". Sources with lower costs will be tried first. (#4619)
- update (#4167)
  - learned to disallow non-fast-forward updates when `ff-only` is given to the `--merge` option.
  - gained a `--follow` option that controls how `--merge` behaves, adding support for merging in the revision that is registered in the parent dataset rather than merging in the configured branch from the sibling.
  - now provides a result record for merge events.
- create-sibling now supports local paths as targets in addition to SSH URLs. (#4187)
- siblings now
  - shows a warning if the caller requests to delete a sibling that does not exist. (#4257)
  - phrases its warning about non-annex repositories in a less alarming way. (#4323)
- The rendering of command errors has been improved. (#4157)
- save now
  - displays a message to signal that the working tree is clean, making it more obvious that no results being rendered corresponds to a clean state. (#4106)
  - provides a stronger warning against using `--to-git`. (#4290)
- diff and save learned about scenarios where they could avoid unnecessary and expensive work. (#4526) (#4544) (#4549)
- Calling diff without `--recursive` but with a path constraint within a subdataset ("/") now traverses into the subdataset, as "/" would, restricting its report to "/". (#4235)
- New option `datalad.annex.retry` controls how many times git-annex will retry on a failed transfer. It defaults to 3 and can be set to 0 to restore the previous behavior. (#4382)
- wtf now warns when the specified dataset does not exist. (#4331)
- The `repr` and `str` output of the dataset and repo classes got a facelift. (#4420) (#4435) (#4439)
- The DataLad Singularity container now comes with p7zip-full.
- DataLad emits a log message when the current working directory is resolved to a different location due to a symlink. This is now logged at the DEBUG rather than WARNING level, as it typically does not indicate a problem. (#4426)
- DataLad now lets the caller know that `git annex init` is scanning for unlocked files, as this operation can be slow in some repositories. (#4316)
- The `log_progress` helper learned how to set the starting point to a non-zero value and how to update the total of an existing progress bar, two features needed for planned improvements to how some commands display their progress. (#4438)

- The `ExternalVersions` object, which is used to check versions of Python modules and external tools (e.g., git-annex), gained an `add` method that enables DataLad extensions and other third-party code to include other programs of interest. (#4441)
- All of the remaining spots that use GitPython have been rewritten without it. Most notably, this includes rewrites of the `clone`, `fetch`, and `push` methods of `GitRepo`. (#4080) (#4087) (#4170) (#4171) (#4175) (#4172)
- When `GitRepo.commit` splits its operation across multiple calls to avoid exceeding the maximum command line length, it now amends to initial commit rather than creating multiple commits. (#4156)
- `GitRepo` gained a `get_corresponding_branch` method (which always returns None), allowing a caller to invoke the method without needing to check if the underlying repo class is `GitRepo` or `AnnexRepo`. (#4274)
- A new helper function `datalad.core.local.repo.repo_from_path` returns a repo class for a specified path. (#4273)
- New `AnnexRepo` method `localsync` performs a `git annex sync` that disables external interaction and is particularly useful for propagating changes on an adjusted branch back to the main branch. (#4243)

1.1.57 0.12.7 (May 22, 2020) – .

Fixes

- Requesting tailored output (`--output=tailored`) from a command with a custom result summary renderer produced repeated output. (#4463)
- A longstanding regression in argcomplete-based command-line completion for Bash has been fixed. You can enable completion by configuring a Bash startup file to run `eval "$(register-python-argcomplete datalad)"` or source DataLad's `tools/cmdline-completion`. The latter should work for Zsh as well. (#4477)
- publish didn't prevent `git-fetch` from recursing into submodules, leading to a failure when the registered submodule was not present locally and the submodule did not have a remote named 'origin'. (#4560)
- addurls botched path handling when the file name format started with "./" and the call was made from a subdirectory of the dataset. (#4504)
- Double dash options in manpages were unintentionally escaped. (#4332)
- The check for HTTP authentication failures crashed in situations where content came in as bytes rather than unicode. (#4543)
- A check in `AnnexRepo.whereis` could lead to a type error. (#4552)
- When installing a dataset to obtain a subdataset, get confusingly displayed a message that described the containing dataset as "underneath" the subdataset. (#4456)
- A couple of Makefile rules didn't properly quote paths. (#4481)
- With DueCredit support enabled (`DUECREDIT_ENABLE=1`), the query for metadata information could flood the output with warnings if datasets didn't have aggregated metadata. The warnings are now silenced, with the overall failure of a metadata call logged at the debug level. (#4568)

Enhancements and new features

- The resource identifier helper learned to recognize URLs with embedded Git transport information, such as gcrypt::https://example.com. (#4529)
- When running non-interactively, a more informative error is now signaled when the UI backend, which cannot display a question, is asked to do so. (#4553)

1.1.58 0.12.6 (April 23, 2020) – .

Major refactoring and deprecations

- The value of `datalad.support.annexrep.N_AUTO_JOBS` is no longer considered. The variable will be removed in a later release. (#4409)

Fixes

- Staring with v0.12.0, `datalad save` recorded the current branch of a parent dataset as the `branch` value in the .gitmodules entry for a subdataset. This behavior is problematic for a few reasons and has been reverted. (#4375)
- The default for the `--jobs` option, "auto", instructed DataLad to pass a value to git-annex's `--jobs` equal to `min(8, max(3, <number of CPUs>))`, which could lead to issues due to the large number of child processes spawned and file descriptors opened. To avoid this behavior, `--jobs=auto` now results in git-annex being called with `--jobs=1` by default. Configure the new option `datalad.runtime.max-annex-jobs` to control the maximum value that will be considered when `--jobs='auto'`. (#4409)
- Various commands have been adjusted to better handle the case where a remote's HEAD ref points to an unborn branch. (#4370)
- search
  - learned to use the query as a regular expression that restricts the keys that are shown for `--show-keys short`. (#4354)
  - gives a more helpful message when query is an invalid regular expression. (#4398)
- The code for parsing Git configuration did not follow Git's behavior of accepting a key with no value as shorthand for key=true. (#4421)
- `AnnexRepo.info` needed a compatibility update for a change in how git-annex reports file names. (#4431)
- create-sibling-github did not gracefully handle a token that did not have the necessary permissions. (#4400)

Enhancements and new features

- search learned to use the query as a regular expression that restricts the keys that are shown for `--show-keys short`. (#4354)
- `datalad <subcommand>` learned to point to the datalad-container extension when a subcommand from that extension is given but the extension is not installed. (#4400) (#4174)

1.1.59 0.12.5 (Apr 02, 2020) – a small step for datalad ...

Fix some bugs and make the world an even better place.

Fixes

- Our `log_progress` helper mishandled the initial display and step of the progress bar. (#4326)
- `AnnexRepo.get_content_annexinfo` is designed to accept `init=None`, but passing that led to an error. (#4330)
- Update a regular expression to handle an output change in Git v2.26.0. (#4328)
- We now set `LC_MESSAGES` to 'C' while running git to avoid failures when parsing output that is marked for translation. (#4342)
- The helper for decoding JSON streams loaded the last line of input without decoding it if the line didn't end with a new line, a regression introduced in the 0.12.0 release. (#4361)
- The clone command failed to git-annex-init a fresh clone whenever it considered to add the origin of the origin as a remote. (#4367)

1.1.60 0.12.4 (Mar 19, 2020) – Windows?!

The main purpose of this release is to have one on PyPi that has no associated wheel to enable a working installation on Windows (#4315).

Fixes

- The description of the `log.outputs` config switch did not keep up with code changes and incorrectly stated that the output would be logged at the DEBUG level; logging actually happens at a lower level. (#4317)

1.1.61 0.12.3 (March 16, 2020) – .

Updates for compatibility with the latest git-annex, along with a few miscellaneous fixes

Major refactoring and deprecations

- All spots that raised a `NoDatasetArgumentFound` exception now raise a `NoDatasetFound` exception to better reflect the situation: it is the *dataset* rather than the *argument* that is not found. For compatibility, the latter inherits from the former, but new code should prefer the latter. (#4285)

Fixes

- Updates for compatibility with git-annex version 8.20200226. (#4214)
- `datalad export-to-figshare` failed to export if the generated title was fewer than three characters. It now queries the caller for the title and guards against titles that are too short. (#4140)
- Authentication was requested multiple times when git-annex launched parallel downloads from the `datalad` special remote. (#4308)

- At verbose logging levels, DataLad requests that git-annex display debugging information too. Work around a bug in git-annex that prevented that from happening. (#4212)
- The internal command runner looked in the wrong place for some configuration variables, including `datalad.log.outputs`, resulting in the default value always being used. (#4194)
- publish failed when trying to publish to a git-lfs special remote for the first time. (#4200)
- `AnnexRepo.set_remote_url` is supposed to establish shared SSH connections but failed to do so. (#4262)

Enhancements and new features

- The message provided when a command cannot determine what dataset to operate on has been improved. (#4285)
- The "aws-s3" authentication type now allows specifying the host through "aws-s3_host", which was needed to work around an authorization error due to a longstanding upstream bug. (#4239)
- The xmp metadata extractor now recognizes ".wav" files.

1.1.62 0.12.2 (Jan 28, 2020) – Smoothen the ride

Mostly a bugfix release with various robustifications, but also makes the first step towards versioned dataset installation requests.

Major refactoring and deprecations

- The minimum required version for GitPython is now 2.1.12. (#4070)

Fixes

- The class for handling configuration values, `ConfigManager`, inappropriately considered the current working directory's dataset, if any, for both reading and writing when instantiated with `dataset=None`. This misbehavior is fairly inaccessible through typical use of DataLad. It affects `datalad.cfg`, the top-level configuration instance that should not consider repository-specific values. It also affects Python users that call `Dataset` with a path that does not yet exist and persists until that dataset is created. (#4078)
- update saved the dataset when called with `--merge`, which is unnecessary and risks committing unrelated changes. (#3996)
- Confusing and irrelevant information about Python defaults have been dropped from the command-line help. (#4002)
- The logic for automatically propagating the 'origin' remote when cloning a local source didn't properly account for relative paths. (#4045)
- Various fixes to file name handling and quoting on Windows. (#4049) (#4050)
- When cloning failed, error lines were not bubbled up to the user in some scenarios. (#4060)

Enhancements and new features

- clone (and thus install)
  - now propagates the `reckless` mode from the superdataset when cloning a dataset into it. (#4037)
  - gained support for `ria+<protocol>://` URLs that point to RIA stores. (#4022)
  - learned to read "@version" from `ria+` URLs and install that version of a dataset (#4036) and to apply URL rewrites configured through Git's `url.*.insteadOf` mechanism (#4064).
  - now copies `datalad.get.subdataset-source-candidate-<name>` options configured within the superdataset into the subdataset. This is particularly useful for RIA data stores. (#4073)
- Archives are now (optionally) handled with 7-Zip instead of `patool`. 7-Zip will be used by default, but `patool` will be used on non-Windows systems if the `datalad.runtime.use-patool` option is set or the `7z` executable is not found. (#4041)

1.1.63 0.12.1 (Jan 15, 2020) – Small bump after big bang

Fix some fallout after major release.

Fixes

- Revert incorrect relative path adjustment to URLs in clone. (#3538)
- Various small fixes to internal helpers and test to run on Windows (#2566) (#2534)

1.1.64 0.12.0 (Jan 11, 2020) – Krakatoa

This release is the result of more than a year of development that includes fixes for a large number of issues, yielding more robust behavior across a wider range of use cases, and introduces major changes in API and behavior. It is the first release for which extensive user documentation is available in a dedicated DataLad Handbook. Python 3 (3.5 and later) is now the only supported Python flavor.

Major changes 0.12 vs 0.11

- save fully replaces add (which is obsolete now, and will be removed in a future release).
- A new Git-annex aware status command enables detailed inspection of dataset hierarchies. The previously available diff command has been adjusted to match status in argument semantics and behavior.
- The ability to configure dataset procedures prior and after the execution of particular commands has been replaced by a flexible "hook" mechanism that is able to run arbitrary DataLad commands whenever command results are detected that match a specification.
- Support of the Windows platform has been improved substantially. While performance and feature coverage on Windows still falls behind Unix-like systems, typical data consumer use cases, and standard dataset operations, such as create and save, are now working. Basic support for data provenance capture via run is also functional.
- Support for Git-annex direct mode repositories has been removed, following the end of support in Git-annex itself.
- The semantics of relative paths in command line arguments have changed. Previously, a call `datalad save --dataset /tmp/myds some/relpath` would have been interpreted as saving a file at `/tmp/myds/some/relpath` into dataset `/tmp/myds`. This has changed to saving `$PWD/some/relpath` into dataset `/tmp/myds`.

More generally, relative paths are now always treated as relative to the current working directory, except for path arguments of Dataset class instance methods of the Python API. The resulting partial duplication of path specifications between path and dataset arguments is mitigated by the introduction of two special symbols that can be given as dataset argument: `^` and `^.`, which identify the topmost superdataset and the closest dataset that contains the working directory, respectively.

- The concept of a "core API" has been introduced. Commands situated in the module `datalad.core` (such as create, save, run, status, diff) receive additional scrutiny regarding API and implementation, and are meant to provide longer-term stability. Application developers are encouraged to preferentially build on these commands.

Major refactoring and deprecations since 0.12.0rc6

- clone has been incorporated into the growing core API. The public `--alternative-source` parameter has been removed, and a `clone_dataset` function with multi-source capabilities is provided instead. The `--reckless` parameter can now take literal mode labels instead of just being a binary flag, but backwards compatibility is maintained.
- The `get_file_content` method of `GitRepo` was no longer used internally or in any known DataLad extensions and has been removed. (#3812)
- The function `get_dataset_root` has been replaced by `rev_get_dataset_root`. `rev_get_dataset_root` remains as a compatibility alias and will be removed in a later release. (#3815)
- The `add_sibling` module, marked obsolete in v0.6.0, has been removed. (#3871)
- `mock` is no longer declared as an external dependency because we can rely on it being in the standard library now that our minimum required Python version is 3.5. (#3860)
- download-url now requires that directories be indicated with a trailing slash rather than interpreting a path as directory when it doesn't exist. This avoids confusion that can result from typos and makes it possible to support directory targets that do not exist. (#3854)
- The `dataset_only` argument of the `ConfigManager` class is deprecated. Use `source="dataset"` instead. (#3907)
- The `--proc-pre` and `--proc-post` options have been removed, and configuration values for `datalad.COMMAND.proc-pre` and `datalad.COMMAND.proc-post` are no longer honored. The new result hook mechanism provides an alternative for `proc-post` procedures. (#3963)

Fixes since 0.12.0rc6

- publish crashed when called with a detached HEAD. It now aborts with an informative message. (#3804)
- Since 0.12.0rc6 the call to update in siblings resulted in a spurious warning. (#3877)
- siblings crashed if it encountered an annex repository that was marked as dead. (#3892)
- The update of rerun in v0.12.0rc3 for the rewritten diff command didn't account for a change in the output of `diff`, leading to `rerun --report` unintentionally including unchanged files in its diff values. (#3873)
- In 0.12.0rc5 download-url was updated to follow the new path handling logic, but its calls to AnnexRepo weren't properly adjusted, resulting in incorrect path handling when the called from a dataset subdirectory. (#3850)
- download-url called `git annex addurl` in a way that failed to register a URL when its header didn't report the content size. (#3911)
- With Git v2.24.0, saving new subdatasets failed due to a bug in that Git release. (#3904)
- With DataLad configured to stop on failure (e.g., specifying `--on-failure=stop` from the command line), a failing result record was not rendered. (#3863)

- Installing a subdataset yielded an "ok" status in cases where the repository was not yet in its final state, making it ineffective for a caller to operate on the repository in response to the result. (#3906)
- The internal helper for converting git-annex's JSON output did not relay information from the "error-messages" field. (#3931)
- run-procedure reported relative paths that were confusingly not relative to the current directory in some cases. It now always reports absolute paths. (#3959)
- diff inappropriately reported files as deleted in some cases when `to` was a value other than `None`. (#3999)
- An assortment of fixes for Windows compatibility. (#3971) (#3974) (#3975) (#3976) (#3979)
- Subdatasets installed from a source given by relative path will now have this relative path used as 'url' in their .gitmodules record, instead of an absolute path generated by Git. (#3538)
- clone will now correctly interpret '~/…' paths as absolute path specifications. (#3958)
- run-procedure mistakenly reported a directory as a procedure. (#3793)
- The cleanup for batched git-annex processes has been improved. (#3794) (#3851)
- The function for adding a version ID to an AWS S3 URL doesn't support URLs with an "s3://" scheme and raises a `NotImplementedError` exception when it encounters one. The function learned to return a URL untouched if an "s3://" URL comes in with a version ID. (#3842)
- A few spots needed to be adjusted for compatibility with git-annex's new `--sameas` feature, which allows special remotes to share a data store. (#3856)
- The `swallow_logs` utility failed to capture some log messages due to an incompatibility with Python 3.7. (#3935)
- siblings
  - crashed if `--inherit` was passed but the parent dataset did not have a remote with a matching name. (#3954)
  - configured the wrong pushurl and annexurl values in some cases. (#3955)

Enhancements and new features since 0.12.0rc6

- By default, datasets cloned from local source paths will now get a configured remote for any recursively discoverable 'origin' sibling that is also available from a local path in order to maximize automatic file availability across local annexes. (#3926)
- The new result hooks mechanism allows callers to specify, via local Git configuration values, DataLad command calls that will be triggered in response to matching result records (i.e., what you see when you call a command with `-f json_pp`). (#3903)
- The command interface classes learned to use a new `_examples_` attribute to render documentation examples for both the Python and command-line API. (#3821)
- Candidate URLs for cloning a submodule can now be generated based on configured templates that have access to various properties of the submodule, including its dataset ID. (#3828)
- DataLad's check that the user's Git identity is configured has been sped up and now considers the appropriate environment variables as well. (#3807)
- The `tag` method of `GitRepo` can now tag revisions other than `HEAD` and accepts a list of arbitrary `git tag` options. (#3787)
- When `get` clones a subdataset and the subdataset's HEAD differs from the commit that is registered in the parent, the active branch of the subdataset is moved to the registered commit if the registered commit is an ancestor of

the subdataset's HEAD commit. This handling has been moved to a more central location within `GitRepo`, and now applies to any `update_submodule(..., init=True)` call. (#3831)

- The output of `datalad -h` has been reformatted to improve readability. (#3862)
- unlock has been sped up. (#3880)
- run-procedure learned to provide and render more information about discovered procedures, including whether the procedure is overridden by another procedure with the same base name. (#3960)
- save now (#3817)
  - records the active branch in the superdataset when registering a new subdataset.
  - calls `git annex sync` when saving a dataset on an adjusted branch so that the changes are brought into the mainline branch.
- subdatasets now aborts when its `dataset` argument points to a non-existent dataset. (#3940)
- wtf now
  - reports the dataset ID if the current working directory is visiting a dataset. (#3888)
  - outputs entries deterministically. (#3927)
- The `ConfigManager` class
  - learned to exclude `.datalad/config` as a source of configuration values, restricting the sources to standard Git configuration files, when called with `source="local"`. (#3907)
  - accepts a value of "override" for its `where` argument to allow Python callers to more convenient override configuration. (#3970)
- Commands now accept a `dataset` value of "^." as shorthand for "the dataset to which the current directory belongs". (#3242)

1.1.65 0.12.0rc6 (Oct 19, 2019) – some releases are better than the others

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

Major refactoring and deprecations

- DataLad no longer supports Python 2. The minimum supported version of Python is now 3.5. (#3629)
- Much of the user-focused content at http://docs.datalad.org has been removed in favor of more up to date and complete material available in the DataLad Handbook. Going forward, the plan is to restrict http://docs.datalad.org to technical documentation geared at developers. (#3678)
- update used to allow the caller to specify which dataset(s) to update as a `PATH` argument or via the the `--dataset` option; now only the latter is supported. Path arguments only serve to restrict which subdataset are updated when operating recursively. (#3700)
- Result records from a get call no longer have a "state" key. (#3746)
- update and get no longer support operating on independent hierarchies of datasets. (#3700) (#3746)
- The run update in 0.12.0rc4 for the new path resolution logic broke the handling of inputs and outputs for calls from a subdirectory. (#3747)
- The `is_submodule_modified` method of `GitRepo` as well as two helper functions in gitrepo.py, `kwargs_to_options` and `split_remote_branch`, were no longer used internally or in any known DataLad extensions and have been removed. (#3702) (#3704)

- The `only_remote` option of `GitRepo.is_with_annex` was not used internally or in any known extensions and has been dropped. (#3768)
- The `get_tags` method of `GitRepo` used to sort tags by committer date. It now sorts them by the tagger date for annotated tags and the committer date for lightweight tags. (#3715)
- The `rev_resolve_path` substituted `resolve_path` helper. (#3797)

Fixes

- Correctly handle relative paths in publish. (#3799) (#3102)
- Do not erroneously discover directory as a procedure. (#3793)
- Correctly extract version from manpage to trigger use of manpages for `--help`. (#3798)
- The `cfg_yoda` procedure saved all modifications in the repository rather than saving only the files it modified. (#3680)
- Some spots in the documentation that were supposed appear as two hyphens were incorrectly rendered in the HTML output en-dashs. (#3692)
- create, install, and clone treated paths as relative to the dataset even when the string form was given, violating the new path handling rules. (#3749) (#3777) (#3780)
- Providing the "^" shortcut to `--dataset` didn't work properly when called from a subdirectory of a subdataset. (#3772)
- We failed to propagate some errors from git-annex when working with its JSON output. (#3751)
- With the Python API, callers are allowed to pass a string or list of strings as the `cfg_proc` argument to create, but the string form was mishandled. (#3761)
- Incorrect command quoting for SSH calls on Windows that rendered basic SSH-related functionality (e.g., sshrun) on Windows unusable. (#3688)
- Annex JSON result handling assumed platform-specific paths on Windows instead of the POSIX-style that is happening across all platforms. (#3719)
- `path_is_under()` was incapable of comparing Windows paths with different drive letters. (#3728)

Enhancements and new features

- Provide a collection of "public" `call_git*` helpers within GitRepo and replace use of "private" and less specific `_git_custom_command` calls. (#3791)
- status gained a `--report-filetype`. Setting it to "raw" can give a performance boost for the price of no longer distinguishing symlinks that point to annexed content from other symlinks. (#3701)
- save disables file type reporting by status to improve performance. (#3712)
- subdatasets (#3743)
  - now extends its result records with a `contains` field that lists which `contains` arguments matched a given subdataset.
  - yields an 'impossible' result record when a `contains` argument wasn't matched to any of the reported subdatasets.
- install now shows more readable output when cloning fails. (#3775)
- `SSHConnection` now displays a more informative error message when it cannot start the `ControlMaster` process. (#3776)

- If the new configuration option `datalad.log.result-level` is set to a single level, all result records will be logged at that level. If you've been bothered by DataLad's double reporting of failures, consider setting this to "debug". (#3754)
- Configuration values from `datalad -c OPTION=VALUE ...` are now validated to provide better errors. (#3695)
- rerun learned how to handle history with merges. As was already the case when cherry picking non-run commits, re-creating merges may results in conflicts, and `rerun` does not yet provide an interface to let the user handle these. (#2754)
- The `fsck` method of `AnnexRepo` has been enhanced to expose more features of the underlying `git fsck` command. (#3693)
- `GitRepo` now has a `for_each_ref_` method that wraps `git for-each-ref`, which is used in various spots that used to rely on GitPython functionality. (#3705)
- Do not pretend to be able to work in optimized (`python -O`) mode, crash early with an informative message. (#3803)

1.1.66 0.12.0rc5 (September 04, 2019) – .

Various fixes and enhancements that bring the 0.12.0 release closer.

Major refactoring and deprecations

- The two modules below have a new home. The old locations still exist as compatibility shims and will be removed in a future release.
  - `datalad.distribution.subdatasets` has been moved to `datalad.local.subdatasets` (#3429)
  - `datalad.interface.run` has been moved to `datalad.core.local.run` (#3444)
- The `lock` method of `AnnexRepo` and the `options` parameter of `AnnexRepo.unlock` were unused internally and have been removed. (#3459)
- The `get_submodules` method of `GitRepo` has been rewritten without GitPython. When the new `compat` flag is true (the current default), the method returns a value that is compatible with the old return value. This backwards-compatible return value and the `compat` flag will be removed in a future release. (#3508)
- The logic for resolving relative paths given to a command has changed (#3435). The new rule is that relative paths are taken as relative to the dataset only if a dataset *instance* is passed by the caller. In all other scenarios they're considered relative to the current directory.

  The main user-visible difference from the command line is that using the `--dataset` argument does *not* result in relative paths being taken as relative to the specified dataset. (The undocumented distinction between "rel/path" and "./rel/path" no longer exists.)

  All commands under `datalad.core` and `datalad.local`, as well as `unlock` and `addurls`, follow the new logic. The goal is for all commands to eventually do so.

Fixes

- The function for loading JSON streams wasn't clever enough to handle content that included a Unicode line separator like U2028. (#3524)
- When unlock was called without an explicit target (i.e., a directory or no paths at all), the call failed if any of the files did not have content present. (#3459)
- `AnnexRepo.get_content_info` failed in the rare case of a key without size information. (#3534)
- save ignored `--on-failure` in its underlying call to status. (#3470)
- Calling remove with a subdirectory displayed spurious warnings about the subdirectory files not existing. (#3586)
- Our processing of `git-annex --json` output mishandled info messages from special remotes. (#3546)
- create
  - didn't bypass the "existing subdataset" check when called with `--force` as of 0.12.0rc3 (#3552)
  - failed to register the up-to-date revision of a subdataset when `--cfg-proc` was used with `--dataset` (#3591)
- The base downloader had some error handling that wasn't compatible with Python 3. (#3622)
- Fixed a number of Unicode py2-compatibility issues. (#3602)
- `AnnexRepo.get_content_annexinfo` did not properly chunk file arguments to avoid exceeding the command-line character limit. (#3587)

Enhancements and new features

- New command `create-sibling-gitlab` provides an interface for creating a publication target on a GitLab instance. (#3447)
- subdatasets (#3429)
  - now supports path-constrained queries in the same manner as commands like `save` and `status`
  - gained a `--contains=PATH` option that can be used to restrict the output to datasets that include a specific path.
  - now narrows the listed subdatasets to those underneath the current directory when called with no arguments
- status learned to accept a plain `--annex` (no value) as shorthand for `--annex basic`. (#3534)
- The `.dirty` property of `GitRepo` and `AnnexRepo` has been sped up. (#3460)
- The `get_content_info` method of `GitRepo`, used by `status` and commands that depend on `status`, now restricts its git calls to a subset of files, if possible, for a performance gain in repositories with many files. (#3508)
- Extensions that do not provide a command, such as those that provide only metadata extractors, are now supported. (#3531)
- When calling git-annex with `--json`, we log standard error at the debug level rather than the warning level if a non-zero exit is expected behavior. (#3518)
- create no longer refuses to create a new dataset in the odd scenario of an empty .git/ directory upstairs. (#3475)
- As of v2.22.0 Git treats a sub-repository on an unborn branch as a repository rather than as a directory. Our documentation and tests have been updated appropriately. (#3476)
- addurls learned to accept a `--cfg-proc` value and pass it to its `create` calls. (#3562)

1.1.67 0.12.0rc4 (May 15, 2019) – the revolution is over

With the replacement of the `save` command implementation with `rev-save` the revolution effort is now over, and the set of key commands for local dataset operations (`create`, `run`, `save`, `status`, `diff`) is now complete. This new core API is available from `datalad.core.local` (and also via `datalad.api`, as any other command).

Major refactoring and deprecations

- The `add` command is now deprecated. It will be removed in a future release.

Fixes

- Remove hard-coded dependencies on POSIX path conventions in SSH support code (#3400)
- Emit an `add` result when adding a new subdataset during save (#3398)
- SSH file transfer now actually opens a shared connection, if none exists yet (#3403)

Enhancements and new features

- `SSHConnection` now offers methods for file upload and download (`get()`, `put()`. The previous `copy()` method only supported upload and was discontinued (#3401)

1.1.68 0.12.0rc3 (May 07, 2019) – the revolution continues

Continues API consolidation and replaces the `create` and `diff` command with more performant implementations.

Major refactoring and deprecations

- The previous `diff` command has been replaced by the diff variant from the datalad-revolution extension. (#3366)
- `rev-create` has been renamed to `create`, and the previous `create` has been removed. (#3383)
- The procedure `setup_yoda_dataset` has been renamed to `cfg_yoda` (#3353).
- The `--nosave` of `addurls` now affects only added content, not newly created subdatasets (#3259).
- `Dataset.get_subdatasets` (deprecated since v0.9.0) has been removed. (#3336)
- The `.is_dirty` method of `GitRepo` and `AnnexRepo` has been replaced by `.status` or, for a subset of cases, the `.dirty` property. (#3330)
- `AnnexRepo.get_status` has been replaced by `AnnexRepo.status`. (#3330)

Fixes

- status
  - reported on directories that contained only ignored files (#3238)
  - gave a confusing failure when called from a subdataset with an explicitly specified dataset argument and "." as a path (#3325)
  - misleadingly claimed that the locally present content size was zero when `--annex basic` was specified (#3378)
- An informative error wasn't given when a download provider was invalid. (#3258)
- Calling `rev-save PATH` saved unspecified untracked subdatasets. (#3288)
- The available choices for command-line options that take values are now displayed more consistently in the help output. (#3326)
- The new pathlib-based code had various encoding issues on Python 2. (#3332)

Enhancements and new features

- wtf now includes information about the Python version. (#3255)
- When operating in an annex repository, checking whether git-annex is available is now delayed until a call to git-annex is actually needed, allowing systems without git-annex to operate on annex repositories in a restricted fashion. (#3274)
- The `load_stream` on helper now supports auto-detection of compressed files. (#3289)
- `create` (formerly `rev-create`)
  - learned to be speedier by passing a path to `status` (#3294)
  - gained a `--cfg-proc` (or `-c`) convenience option for running configuration procedures (or more accurately any procedure that begins with "cfg_") in the newly created dataset (#3353)
- `AnnexRepo.set_metadata` now returns a list while `AnnexRepo.set_metadata_` returns a generator, a behavior which is consistent with the `add` and `add_` method pair. (#3298)
- `AnnexRepo.get_metadata` now supports batch querying of known annex files. Note, however, that callers should carefully validate the input paths because the batch call will silently hang if given non-annex files. (#3364)
- status
  - now reports a "bytesize" field for files tracked by Git (#3299)
  - gained a new option `eval_subdataset_state` that controls how the subdataset state is evaluated. Depending on the information you need, you can select a less expensive mode to make `status` faster. (#3324)
  - colors deleted files "red" (#3334)
- Querying repository content is faster due to batching of `git cat-file` calls. (#3301)
- The dataset ID of a subdataset is now recorded in the superdataset. (#3304)
- `GitRepo.diffstatus`
  - now avoids subdataset recursion when the comparison is not with the working tree, which substantially improves performance when diffing large dataset hierarchies (#3314)
  - got smarter and faster about labeling a subdataset as "modified" (#3343)

- `GitRepo.get_content_info` now supports disabling the file type evaluation, which gives a performance boost in cases where this information isn't needed. (#3362)
- The XMP metadata extractor now filters based on file name to improve its performance. (#3329)

1.1.69 0.12.0rc2 (Mar 18, 2019) – revolution!

Fixes

- `GitRepo.dirty` does not report on nested empty directories (#3196).
- `GitRepo.save()` reports results on deleted files.

Enhancements and new features

- Absorb a new set of core commands from the datalad-revolution extension:
  - `rev-status`: like `git status`, but simpler and working with dataset hierarchies
  - `rev-save`: a 2-in-1 replacement for save and add
  - `rev-create`: a ~30% faster create
- JSON support tools can now read and write compressed files.

1.1.70 0.12.0rc1 (Mar 03, 2019) – to boldly go ...

Major refactoring and deprecations

- Discontinued support for git-annex direct-mode (also no longer supported upstream).

Enhancements and new features

- Dataset and Repo object instances are now hashable, and can be created based on pathlib Path object instances
- Imported various additional methods for the Repo classes to query information and save changes.

1.1.71 0.11.8 (Oct 11, 2019) – annex-we-are-catching-up

Fixes

- Our internal command runner failed to capture output in some cases. (#3656)
- Workaround in the tests around python in cPython >= 3.7.5 ';' in the filename confusing mimetypes (#3769) (#3770)

Enhancements and new features

- Prepared for upstream changes in git-annex, including support for the latest git-annex
  - 7.20190912 auto-upgrades v5 repositories to v7. (#3648) (#3682)
  - 7.20191009 fixed treatment of (larger/smaller)than in .gitattributes (#3765)
- The `cfg_text2git` procedure, as well the `--text-no-annex` option of create, now configure .gitattributes so that empty files are stored in git rather than annex. (#3667)

1.1.72 0.11.7 (Sep 06, 2019) – python2-we-still-love-you-but-...

Primarily bugfixes with some optimizations and refactorings.

Fixes

- addurls
  - now provides better handling when the URL file isn't in the expected format. (#3579)
  - always considered a relative file for the URL file argument as relative to the current working directory, which goes against the convention used by other commands of taking relative paths as relative to the dataset argument. (#3582)
- run-procedure
  - hard coded "python" when formatting the command for non-executable procedures ending with ".py". `sys.executable` is now used. (#3624)
  - failed if arguments needed more complicated quoting than simply surrounding the value with double quotes. This has been resolved for systems that support `shlex.quote`, but note that on Windows values are left unquoted. (#3626)
- siblings now displays an informative error message if a local path is given to `--url` but `--name` isn't specified. (#3555)
- sshrun, the command DataLad uses for `GIT_SSH_COMMAND`, didn't support all the parameters that Git expects it to. (#3616)
- Fixed a number of Unicode py2-compatibility issues. (#3597)
- download-url now will create leading directories of the output path if they do not exist (#3646)

Enhancements and new features

- The annotate-paths helper now caches subdatasets it has seen to avoid unnecessary calls. (#3570)
- A repeated configuration query has been dropped from the handling of `--proc-pre` and `--proc-post`. (#3576)
- Calls to `git annex find` now use `--in=.` instead of the alias `--in=here` to take advantage of an optimization that git-annex (as of the current release, 7.20190730) applies only to the former. (#3574)
- addurls now suggests close matches when the URL or file format contains an unknown field. (#3594)
- Shared logic used in the setup.py files of DataLad and its extensions has been moved to modules in the _datalad_build_support/ directory. (#3600)
- Get ready for upcoming git-annex dropping support for direct mode (#3631)

1.1.73 0.11.6 (Jul 30, 2019) – am I the last of 0.11.x?

Primarily bug fixes to achieve more robust performance

Fixes

- Our tests needed various adjustments to keep up with upstream changes in Travis and Git. (#3479) (#3492) (#3493)
- `AnnexRepo.is_special_annex_remote` was too selective in what it considered to be a special remote. (#3499)
- We now provide information about unexpected output when git-annex is called with `--json`. (#3516)
- Exception logging in the `__del__` method of `GitRepo` and `AnnexRepo` no longer fails if the names it needs are no longer bound. (#3527)
- addurls botched the construction of subdataset paths that were more than two levels deep and failed to create datasets in a reliable, breadth-first order. (#3561)
- Cloning a `type=git` special remote showed a spurious warning about the remote not being enabled. (#3547)

Enhancements and new features

- For calls to git and git-annex, we disable automatic garbage collection due to past issues with GitPython's state becoming stale, but doing so results in a larger .git/objects/ directory that isn't cleaned up until garbage collection is triggered outside of DataLad. Tests with the latest GitPython didn't reveal any state issues, so we've re-enabled automatic garbage collection. (#3458)
- rerun learned an `--explicit` flag, which it relays to its calls to [run][[]]. This makes it possible to call `rerun` in a dirty working tree (#3498).
- The metadata command aborts earlier if a metadata extractor is unavailable. (#3525)

1.1.74 0.11.5 (May 23, 2019) – stability is not overrated

Should be faster and less buggy, with a few enhancements.

Fixes

- create-sibling (#3318)
  - Siblings are no longer configured with a post-update hook unless a web interface is requested with `--ui`.
  - `git submodule update --init` is no longer called from the post-update hook.
  - If `--inherit` is given for a dataset without a superdataset, a warning is now given instead of raising an error.
- The internal command runner failed on Python 2 when its `env` argument had unicode values. (#3332)
- The safeguard that prevents creating a dataset in a subdirectory that already contains tracked files for another repository failed on Git versions before 2.14. For older Git versions, we now warn the caller that the safeguard is not active. (#3347)
- A regression introduced in v0.11.1 prevented save from committing changes under a subdirectory when the subdirectory was specified as a path argument. (#3106)

- A workaround introduced in v0.11.1 made it possible for save to do a partial commit with an annex file that has gone below the `annex.largefiles` threshold. The logic of this workaround was faulty, leading to files being displayed as typechanged in the index following the commit. (#3365)
- The resolve_path() helper confused paths that had a semicolon for SSH RIs. (#3425)
- The detection of SSH RIs has been improved. (#3425)

Enhancements and new features

- The internal command runner was too aggressive in its decision to sleep. (#3322)
- The "INFO" label in log messages now retains the default text color for the terminal rather than using white, which only worked well for terminals with dark backgrounds. (#3334)
- A short flag `-R` is now available for the `--recursion-limit` flag, a flag shared by several subcommands. (#3340)
- The authentication logic for create-sibling-github has been revamped and now supports 2FA. (#3180)
- New configuration option `datalad.ui.progressbar` can be used to configure the default backend for progress reporting ("none", for example, results in no progress bars being shown). (#3396)
- A new progress backend, available by setting datalad.ui.progressbar to "log", replaces progress bars with a log message upon completion of an action. (#3396)
- DataLad learned to consult the NO_COLOR environment variable and the new `datalad.ui.color` configuration option when deciding to color output. The default value, "auto", retains the current behavior of coloring output if attached to a TTY (#3407).
- clean now removes annex transfer directories, which is useful for cleaning up failed downloads. (#3374)
- clone no longer refuses to clone into a local path that looks like a URL, making its behavior consistent with `git clone`. (#3425)
- wtf
  - Learned to fall back to the `dist` package if `platform.dist`, which has been removed in the yet-to-be-release Python 3.8, does not exist. (#3439)
  - Gained a `--section` option for limiting the output to specific sections and a `--decor` option, which currently knows how to format the output as GitHub's `<details>` section. (#3440)

1.1.75 0.11.4 (Mar 18, 2019) – get-ready

Largely a bug fix release with a few enhancements

Important

- 0.11.x series will be the last one with support for direct mode of git-annex which is used on crippled (no symlinks and no locking) filesystems. v7 repositories should be used instead.

Fixes

- Extraction of .gz files is broken without p7zip installed. We now abort with an informative error in this situation. (#3176)
- Committing failed in some cases because we didn't ensure that the path passed to `git read-tree --index-output=...` resided on the same filesystem as the repository. (#3181)
- Some pointless warnings during metadata aggregation have been eliminated. (#3186)
- With Python 3 the LORIS token authenticator did not properly decode a response (#3205).
- With Python 3 downloaders unnecessarily decoded the response when getting the status, leading to an encoding error. (#3210)
- In some cases, our internal command Runner did not adjust the environment's `PWD` to match the current working directory specified with the `cwd` parameter. (#3215)
- The specification of the pyliblzma dependency was broken. (#3220)
- search displayed an uninformative blank log message in some cases. (#3222)
- The logic for finding the location of the aggregate metadata DB anchored the search path incorrectly, leading to a spurious warning. (#3241)
- Some progress bars were still displayed when stdout and stderr were not attached to a tty. (#3281)
- Check for stdin/out/err to not be closed before checking for `.isatty`. (#3268)

Enhancements and new features

- Creating a new repository now aborts if any of the files in the directory are tracked by a repository in a parent directory. (#3211)
- run learned to replace the `{tmpdir}` placeholder in commands with a temporary directory. (#3223)
- duecredit support has been added for citing DataLad itself as well as datasets that an analysis uses. (#3184)
- The `eval_results` interface helper unintentionally modified one of its arguments. (#3249)
- A few DataLad constants have been added, changed, or renamed (#3250):
  - `HANDLE_META_DIR` is now `DATALAD_DOTDIR`. The old name should be considered deprecated.
  - `METADATA_DIR` now refers to `DATALAD_DOTDIR/metadata` rather than `DATALAD_DOTDIR/meta` (which is still available as `OLDMETADATA_DIR`).
  - The new `DATASET_METADATA_FILE` refers to `METADATA_DIR/dataset.json`.
  - The new `DATASET_CONFIG_FILE` refers to `DATALAD_DOTDIR/config`.
  - `METADATA_FILENAME` has been renamed to `OLDMETADATA_FILENAME`.

1.1.76 0.11.3 (Feb 19, 2019) – read-me-gently

Just a few of important fixes and minor enhancements.

Fixes

- The logic for setting the maximum command line length now works around Python 3.4 returning an unreasonably high value for `SC_ARG_MAX` on Debian systems. (#3165)
- DataLad commands that are conceptually "read-only", such as `datalad ls -L`, can fail when the caller lacks write permissions because git-annex tries merging remote git-annex branches to update information about availability. DataLad now disables `annex.merge-annex-branches` in some common "read-only" scenarios to avoid these failures. (#3164)

Enhancements and new features

- Accessing an "unbound" dataset method now automatically imports the necessary module rather than requiring an explicit import from the Python caller. For example, calling `Dataset.add` no longer needs to be preceded by `from datalad.distribution.add import Add` or an import of `datalad.api`. (#3156)
- Configuring the new variable `datalad.ssh.identityfile` instructs DataLad to pass a value to the `-i` option of `ssh`. (#3149) (#3168)

1.1.77 0.11.2 (Feb 07, 2019) – live-long-and-prosper

A variety of bugfixes and enhancements

Major refactoring and deprecations

- All extracted metadata is now placed under git-annex by default. Previously files smaller than 20 kb were stored in git. (#3109)
- The function `datalad.cmd.get_runner` has been removed. (#3104)

Fixes

- Improved handling of long commands:
  - The code that inspected `SC_ARG_MAX` didn't check that the reported value was a sensible, positive number. (#3025)
  - More commands that invoke `git` and `git-annex` with file arguments learned to split up the command calls when it is likely that the command would fail due to exceeding the maximum supported length. (#3138)
- The `setup_yoda_dataset` procedure created a malformed .gitattributes line. (#3057)
- download-url unnecessarily tried to infer the dataset when `--no-save` was given. (#3029)
- rerun aborted too late and with a confusing message when a ref specified via `--onto` didn't exist. (#3019)
- run:
  - `run` didn't preserve the current directory prefix ("./") on inputs and outputs, which is problematic if the caller relies on this representation when formatting the command. (#3037)
  - Fixed a number of unicode py2-compatibility issues. (#3035) (#3046)

  - To proceed with a failed command, the user was confusingly instructed to use `save` instead of `add` even though `run` uses `add` underneath. (#3080)

- Fixed a case where the helper class for checking external modules incorrectly reported a module as unknown. (#3051)
- add-archive-content mishandled the archive path when the leading path contained a symlink. (#3058)
- Following denied access, the credential code failed to consider a scenario, leading to a type error rather than an appropriate error message. (#3091)
- Some tests failed when executed from a `git worktree` checkout of the source repository. (#3129)
- During metadata extraction, batched annex processes weren't properly terminated, leading to issues on Windows. (#3137)
- add incorrectly handled an "invalid repository" exception when trying to add a submodule. (#3141)
- Pass `GIT_SSH_VARIANT=ssh` to git processes to be able to specify alternative ports in SSH urls

Enhancements and new features

- search learned to suggest closely matching keys if there are no hits. (#3089)
- create-sibling
  - gained a `--group` option so that the caller can specify the file system group for the repository. (#3098)
  - now understands SSH URLs that have a port in them (i.e. the "ssh://{[}user@{]}host.xz{[}:port{]}/path/to/repo.git/" syntax mentioned in `man git-fetch`). (#3146)
- Interface classes can now override the default renderer for summarizing results. (#3061)
- run:
  - `--input` and `--output` can now be shortened to `-i` and `-o`. (#3066)
  - Placeholders such as "{inputs}" are now expanded in the command that is shown in the commit message subject. (#3065)
  - `interface.run.run_command` gained an `extra_inputs` argument so that wrappers like datalad-container can specify additional inputs that aren't considered when formatting the command string. (#3038)
  - "–" can now be used to separate options for `run` and those for the command in ambiguous cases. (#3119)
- The utilities `create_tree` and `ok_file_has_content` now support ".gz" files. (#3049)
- The Singularity container for 0.11.1 now uses nd_freeze to make its builds reproducible.
- A publications page has been added to the documentation. (#3099)
- `GitRepo.set_gitattributes` now accepts a `mode` argument that controls whether the .gitattributes file is appended to (default) or overwritten. (#3115)
- `datalad --help` now avoids using `man` so that the list of subcommands is shown. (#3124)

1.1.78 0.11.1 (Nov 26, 2018) – v7-better-than-v6

Rushed out bugfix release to stay fully compatible with recent git-annex which introduced v7 to replace v6.

Fixes

- install: be able to install recursively into a dataset (#2982)
- save: be able to commit/save changes whenever files potentially could have swapped their storage between git and annex (#1651) (#2752) (#3009)
- [aggregate-metadata][]:
  - dataset's itself is now not "aggregated" if specific paths are provided for aggregation (#3002). That resolves the issue of `-r` invocation aggregating all subdatasets of the specified dataset as well
  - also compare/verify the actual content checksum of aggregated metadata while considering subdataset metadata for re-aggregation (#3007)
- `annex` commands are now chunked assuming 50% "safety margin" on the maximal command line length. Should resolve crashes while operating of too many files at ones (#3001)
- `run` sidecar config processing (#2991)
- no double trailing period in docs (#2984)
- correct identification of the repository with symlinks in the paths in the tests (#2972)
- re-evaluation of dataset properties in case of dataset changes (#2946)
- [text2git][] procedure to use `ds.repo.set_gitattributes` (#2974) (#2954)
- Switch to use plain `os.getcwd()` if inconsistency with env var `$PWD` is detected (#2914)
- Make sure that credential defined in env var takes precedence (#2960) (#2950)

Enhancements and new features

- shub://datalad/datalad:git-annex-dev provides a Debian buster Singularity image with build environment for git-annex. `tools/bisect-git-annex` provides a helper for running `git bisect` on git-annex using that Singularity container (#2995)
- Added `.zenodo.json` for better integration with Zenodo for citation
- run-procedure now provides names and help messages with a custom renderer for (#2993)
- Documentation: point to datalad-revolution extension (prototype of the greater DataLad future)
- run
  - support injecting of a detached command (#2937)
- `annex` metadata extractor now extracts `annex.key` metadata record. Should allow now to identify uses of specific files etc (#2952)
- Test that we can install from http://datasets.datalad.org
- Proper rendering of `CommandError` (e.g. in case of "out of space" error) (#2958)

1.1.79 0.11.0 (Oct 23, 2018) – Soon-to-be-perfect

git-annex 6.20180913 (or later) is now required - provides a number of fixes for v6 mode operations etc.

Major refactoring and deprecations

- `datalad.consts.LOCAL_CENTRAL_PATH` constant was deprecated in favor of `datalad.locations.default-dataset` configuration variable (#2835)

Minor refactoring

- `"notneeded"` messages are no longer reported by default results renderer
- run no longer shows commit instructions upon command failure when `explicit` is true and no outputs are specified (#2922)
- `get_git_dir` moved into GitRepo (#2886)
- `_gitpy_custom_call` removed from GitRepo (#2894)
- `GitRepo.get_merge_base` argument is now called `commitishes` instead of `treeishes` (#2903)

Fixes

- update should not leave the dataset in non-clean state (#2858) and some other enhancements (#2859)
- Fixed chunking of the long command lines to account for decorators and other arguments (#2864)
- Progress bar should not crash the process on some missing progress information (#2891)
- Default value for `jobs` set to be `"auto"` (not `None`) to take advantage of possible parallel get if in `-g` mode (#2861)
- wtf must not crash if `git-annex` is not installed etc (#2865), (#2865), (#2918), (#2917)
- Fixed paths (with spaces etc) handling while reporting annex error output (#2892), (#2893)
- `__del__` should not access `.repo` but `._repo` to avoid attempts for reinstantiation etc (#2901)
- Fix up submodule `.git` right in `GitRepo.add_submodule` to avoid added submodules being non git-annex friendly (#2909), (#2904)
- run-procedure (#2905)
  - now will provide dataset into the procedure if called within dataset
  - will not crash if procedure is an executable without `.py` or `.sh` suffixes
- Use centralized `.gitattributes` handling while setting annex backend (#2912)
- `GlobbedPaths.expand(..., full=True)` incorrectly returned relative paths when called more than once (#2921)

Enhancements and new features

- Report progress on clone when installing from "smart" git servers (#2876)
- Stale/unused `sth_like_file_has_content` was removed (#2860)
- Enhancements to search to operate on "improved" metadata layouts (#2878)
- Output of `git annex init` operation is now logged (#2881)
- New
  - `GitRepo.cherry_pick` (#2900)
  - `GitRepo.format_commit` (#2902)
- run-procedure (#2905)
  - procedures can now recursively be discovered in subdatasets as well. The uppermost has highest priority
  - Procedures in user and system locations now take precedence over those in datasets.

1.1.80 0.10.3.1 (Sep 13, 2018) – Nothing-is-perfect

Emergency bugfix to address forgotten boost of version in `datalad/version.py`.

1.1.81 0.10.3 (Sep 13, 2018) – Almost-perfect

This is largely a bugfix release which addressed many (but not yet all) issues of working with git-annex direct and version 6 modes, and operation on Windows in general. Among enhancements you will see the support of public S3 buckets (even with periods in their names), ability to configure new providers interactively, and improved `egrep` search backend.

Although we do not require with this release, it is recommended to make sure that you are using a recent `git-annex` since it also had a variety of fixes and enhancements in the past months.

Fixes

- Parsing of combined short options has been broken since DataLad v0.10.0. (#2710)
- The `datalad save` instructions shown by `datalad run` for a command with a non-zero exit were incorrectly formatted. (#2692)
- Decompression of zip files (e.g., through `datalad add-archive-content`) failed on Python 3. (#2702)
- Windows:
  - colored log output was not being processed by colorama. (#2707)
  - more codepaths now try multiple times when removing a file to deal with latency and locking issues on Windows. (#2795)
- Internal git fetch calls have been updated to work around a GitPython `BadName` issue. (#2712), (#2794)
- The progress bar for annex file transferring was unable to handle an empty file. (#2717)
- `datalad add-readme` halted when no aggregated metadata was found rather than displaying a warning. (#2731)
- `datalad rerun` failed if `--onto` was specified and the history contained no run commits. (#2761)

- Processing of a command's results failed on a result record with a missing value (e.g., absent field or subfield in metadata). Now the missing value is rendered as "N/A". (#2725).
- A couple of documentation links in the "Delineation from related solutions" were misformatted. (#2773)
- With the latest git-annex, several known V6 failures are no longer an issue. (#2777)
- In direct mode, commit changes would often commit annexed content as regular Git files. A new approach fixes this and resolves a good number of known failures. (#2770)
- The reporting of command results failed if the current working directory was removed (e.g., after an unsuccessful `install`). (#2788)
- When installing into an existing empty directory, `datalad install` removed the directory after a failed clone. (#2788)
- `datalad run` incorrectly handled inputs and outputs for paths with spaces and other characters that require shell escaping. (#2798)
- Globbing inputs and outputs for `datalad run` didn't work correctly if a subdataset wasn't installed. (#2796)
- Minor (in)compatibility with git 2.19 - (no) trailing period in an error message now. (#2815)

Enhancements and new features

- Anonymous access is now supported for S3 and other downloaders. (#2708)
- A new interface is available to ease setting up new providers. (#2708)
- Metadata: changes to egrep mode search (#2735)
  - Queries in egrep mode are now case-sensitive when the query contains any uppercase letters and are case-insensitive otherwise. The new mode egrepcs can be used to perform a case-sensitive query with all lower-case letters.
  - Search can now be limited to a specific key.
  - Multiple queries (list of expressions) are evaluated using AND to determine whether something is a hit.
  - A single multi-field query (e.g., `pa*:findme`) is a hit, when any matching field matches the query.
  - All matching key/value combinations across all (multi-field) queries are reported in the query_matched result field.
  - egrep mode now shows all hits rather than limiting the results to the top 20 hits.
- The documentation on how to format commands for `datalad run` has been improved. (#2703)
- The method for determining the current working directory on Windows has been improved. (#2707)
- `datalad --version` now simply shows the version without the license. (#2733)
- `datalad export-archive` learned to export under an existing directory via its `--filename` option. (#2723)
- `datalad export-to-figshare` now generates the zip archive in the root of the dataset unless `--filename` is specified. (#2723)
- After importing `datalad.api`, `help(datalad.api)` (or `datalad.api?` in IPython) now shows a summary of the available DataLad commands. (#2728)
- Support for using `datalad` from IPython has been improved. (#2722)
- `datalad wtf` now returns structured data and reports the version of each extension. (#2741)
- The internal handling of gitattributes information has been improved. A user-visible consequence is that `datalad create --force` no longer duplicates existing attributes. (#2744)

- The "annex" metadata extractor can now be used even when no content is present. (#2724)
- The `add_url_to_file` method (called by commands like `datalad download-url` and `datalad add-archive-content`) learned how to display a progress bar. (#2738)

1.1.82 0.10.2 (Jul 09, 2018) – Thesecuriestever

Primarily a bugfix release to accommodate recent git-annex release forbidding file:// and http://localhost/ URLs which might lead to revealing private files if annex is publicly shared.

Fixes

- fixed testing to be compatible with recent git-annex (6.20180626)
- download-url will now download to current directory instead of the top of the dataset

Enhancements and new features

- do not quote ~ in URLs to be consistent with quote implementation in Python 3.7 which now follows RFC 3986
- run support for user-configured placeholder values
- documentation on native git-annex metadata support
- handle 401 errors from LORIS tokens
- `yoda` procedure will instantiate `README.md`
- `--discover` option added to run-procedure to list available procedures

1.1.83 0.10.1 (Jun 17, 2018) – OHBM polish

The is a minor bugfix release.

Fixes

- Be able to use backports.lzma as a drop-in replacement for pyliblzma.
- Give help when not specifying a procedure name in `run-procedure`.
- Abort early when a downloader received no filename.
- Avoid `rerun` error when trying to unlock non-available files.

1.1.84 0.10.0 (Jun 09, 2018) – The Release

This release is a major leap forward in metadata support.

Major refactoring and deprecations

- Metadata
  - Prior metadata provided by datasets under `.datalad/meta` is no longer used or supported. Metadata must be reaggregated using 0.10 version
  - Metadata extractor types are no longer auto-guessed and must be explicitly specified in `datalad.metadata.nativetype` config (could contain multiple values)
  - Metadata aggregation of a dataset hierarchy no longer updates all datasets in the tree with new metadata. Instead, only the target dataset is updated. This behavior can be changed via the –update-mode switch. The new default prevents needless modification of (3rd-party) subdatasets.
  - Neuroimaging metadata support has been moved into a dedicated extension: https://github.com/datalad/datalad-neuroimaging
- Crawler
  - moved into a dedicated extension: https://github.com/datalad/datalad-crawler
- `export_tarball` plugin has been generalized to `export_archive` and can now also generate ZIP archives.
- By default a dataset X is now only considered to be a super-dataset of another dataset Y, if Y is also a registered subdataset of X.

Fixes

A number of fixes did not make it into the 0.9.x series:

- Dynamic configuration overrides via the `-c` option were not in effect.
- `save` is now more robust with respect to invocation in subdirectories of a dataset.
- `unlock` now reports correct paths when running in a dataset subdirectory.
- `get` is more robust to path that contain symbolic links.
- symlinks to subdatasets of a dataset are now correctly treated as a symlink, and not as a subdataset
- `add` now correctly saves staged subdataset additions.
- Running `datalad save` in a dataset no longer adds untracked content to the dataset. In order to add content a path has to be given, e.g. `datalad save .`
- `wtf` now works reliably with a DataLad that wasn't installed from Git (but, e.g., via pip)
- More robust URL handling in `simple_with_archives` crawler pipeline.

Enhancements and new features

- Support for DataLad extension that can contribute API components from 3rd-party sources, incl. commands, metadata extractors, and test case implementations. See https://github.com/datalad/datalad-extension-template for a demo extension.
- Metadata (everything has changed!)
  - Metadata extraction and aggregation is now supported for datasets and individual files.
  - Metadata query via `search` can now discover individual files.

  - Extracted metadata can now be stored in XZ compressed files, is optionally annexed (when exceeding a configurable size threshold), and obtained on demand (new configuration option `datalad.metadata.create-aggregate-annex-limit`).
  - Status and availability of aggregated metadata can now be reported via `metadata --get-aggregates`
  - New configuration option `datalad.metadata.maxfieldsize` to exclude too large metadata fields from aggregation.
  - The type of metadata is no longer guessed during metadata extraction. A new configuration option `datalad.metadata.nativetype` was introduced to enable one or more particular metadata extractors for a dataset.
  - New configuration option `datalad.metadata.store-aggregate-content` to enable the storage of aggregated metadata for dataset content (i.e. file-based metadata) in contrast to just metadata describing a dataset as a whole.
- `search` was completely reimplemented. It offers three different modes now:
  - 'egrep' (default): expression matching in a plain string version of metadata
  - 'textblob': search a text version of all metadata using a fully featured query language (fast indexing, good for keyword search)
  - 'autofield': search an auto-generated index that preserves individual fields of metadata that can be represented in a tabular structure (substantial indexing cost, enables the most detailed queries of all modes)
- New extensions:
  - addurls, an extension for creating a dataset (and possibly subdatasets) from a list of URLs.
  - export_to_figshare
  - extract_metadata
- add_readme makes use of available metadata
- By default the wtf extension now hides sensitive information, which can be included in the output by passing `--senstive=some` or `--senstive=all`.
- Reduced startup latency by only importing commands necessary for a particular command line call.
- create:
  - `-d <parent> --nosave` now registers subdatasets, when possible.
  - `--fake-dates` configures dataset to use fake-dates
- run now provides a way for the caller to save the result when a command has a non-zero exit status.
- `datalad rerun` now has a `--script` option that can be used to extract previous commands into a file.
- A DataLad Singularity container is now available on Singularity Hub.
- More casts have been embedded in the use case section of the documentation.
- `datalad --report-status` has a new value 'all' that can be used to temporarily re-enable reporting that was disable by configuration settings.

1.1.85 0.9.3 (Mar 16, 2018) – pi+0.02 release

Some important bug fixes which should improve usability

Fixes

- `datalad-archives` special remote now will lock on acquiring or extracting an archive - this allows for it to be used with -J flag for parallel operation
- relax introduced in 0.9.2 demand on git being configured for datalad operation - now we will just issue a warning
- `datalad ls` should now list "authored date" and work also for datasets in detached HEAD mode
- `datalad save` will now save original file as well, if file was "git mv"ed, so you can now `datalad run git mv old new` and have changes recorded

Enhancements and new features

- `--jobs` argument now could take `auto` value which would decide on # of jobs depending on the # of available CPUs. `git-annex` > 6.20180314 is recommended to avoid regression with -J.
- memoize calls to `RI` meta-constructor – should speed up operation a bit
- `DATALAD_SEED` environment variable could be used to seed Python RNG and provide reproducible UUIDs etc (useful for testing and demos)

1.1.86 0.9.2 (Mar 04, 2018) – it is (again) better than ever

Largely a bugfix release with a few enhancements.

Fixes

- Execution of external commands (git) should not get stuck when lots of both stdout and stderr output, and should not loose remaining output in some cases
- Config overrides provided in the command line (-c) should now be handled correctly
- Consider more remotes (not just tracking one, which might be none) while installing subdatasets
- Compatibility with git 2.16 with some changed behaviors/annotations for submodules
- Fail `remove` if `annex drop` failed
- Do not fail operating on files which start with dash (-)
- URL unquote paths within S3, URLs and DataLad RIs (///)
- In non-interactive mode fail if authentication/access fails
- Web UI:
  - refactored a little to fix incorrect listing of submodules in subdirectories
  - now auto-focuses on search edit box upon entering the page
- Assure that extracted from tarballs directories have executable bit set

Enhancements and new features

- A log message and progress bar will now inform if a tarball to be downloaded while getting specific files (requires git-annex > 6.20180206)
- A dedicated `datalad rerun` command capable of rerunning entire sequences of previously `run` commands. **Reproducibility through VCS. Use ``run`` even if not interested in ``rerun``**
- Alert the user if `git` is not yet configured but git operations are requested
- Delay collection of previous ssh connections until it is actually needed. Also do not require ':' while specifying ssh host
- AutomagicIO: Added proxying of isfile, lzma.LZMAFile and io.open
- Testing:
  - added DATALAD_DATASETS_TOPURL=http://datasets-tests.datalad.org to run tests against another website to not obscure access stats
  - tests run against temporary HOME to avoid side-effects
  - better unit-testing of interactions with special remotes
- CONTRIBUTING.md describes how to setup and use `git-hub` tool to "attach" commits to an issue making it into a PR
- DATALAD_USE_DEFAULT_GIT env variable could be used to cause DataLad to use default (not the one possibly bundled with git-annex) git
- Be more robust while handling not supported requests by annex in special remotes
- Use of `swallow_logs` in the code was refactored away – less mysteries now, just increase logging level
- `wtf` plugin will report more information about environment, externals and the system

1.1.87 0.9.1 (Oct 01, 2017) – "DATALAD!"(JBTM)

Minor bugfix release

Fixes

- Should work correctly with subdatasets named as numbers of bool values (requires also GitPython >= 2.1.6)
- Custom special remotes should work without crashing with git-annex >= 6.20170924

1.1.88 0.9.0 (Sep 19, 2017) – isn't it a lucky day even though not a Friday?

Major refactoring and deprecations

- the `files` argument of save has been renamed to `path` to be uniform with any other command
- all major commands now implement more uniform API semantics and result reporting. Functionality for modification detection of dataset content has been completely replaced with a more efficient implementation
- publish now features a `--transfer-data` switch that allows for a disambiguous specification of whether to publish data – independent of the selection which datasets to publish (which is done via their paths). Moreover, publish now transfers data before repository content is pushed.

Fixes

- drop no longer errors when some subdatasets are not installed
- install will no longer report nothing when a Dataset instance was given as a source argument, but rather perform as expected
- remove doesn't remove when some files of a dataset could not be dropped
- publish
  - no longer hides error during a repository push
  - publish behaves "correctly" for `--since=` in considering only the differences the last "pushed" state
  - data transfer handling while publishing with dependencies, to github
- improved robustness with broken Git configuration
- search should search for unicode strings correctly and not crash
- robustify git-annex special remotes protocol handling to allow for spaces in the last argument
- UI credentials interface should now allow to Ctrl-C the entry
- should not fail while operating on submodules named with numerics only or by bool (true/false) names
- crawl templates should not now override settings for `largefiles` if specified in `.gitattributes`

Enhancements and new features

- **Exciting new feature** run command to protocol execution of an external command and rerun computation if desired. See screencast
- save now uses Git for detecting with sundatasets need to be inspected for potential changes, instead of performing a complete traversal of a dataset tree
- add looks for changes relative to the last committed state of a dataset to discover files to add more efficiently
- diff can now report untracked files in addition to modified files
- [uninstall][] will check itself whether a subdataset is properly registered in a superdataset, even when no superdataset is given in a call
- subdatasets can now configure subdatasets for exclusion from recursive installation (`datalad-recursiveinstall` submodule configuration property)
- precrafted pipelines of [crawl][] now will not override `annex.largefiles` setting if any was set within `.gitattribues` (e.g. by `datalad create --text-no-annex`)
- framework for screencasts: `tools/cast*` tools and sample cast scripts under `doc/casts` which are published at datalad.org/features.html
- new project YouTube channel
- tests failing in direct and/or v6 modes marked explicitly

1.1.89 0.8.1 (Aug 13, 2017) – the best birthday gift

Bugfixes

Fixes

- Do not attempt to update a not installed sub-dataset
- In case of too many files to be specified for get or copy_to, we will make multiple invocations of underlying git-annex command to not overfill command line
- More robust handling of unicode output in terminals which might not support it

Enhancements and new features

- Ship a copy of numpy.testing to facilitate [test][] without requiring numpy as dependency. Also allow to pass to command which test(s) to run
- In get and copy_to provide actual original requested paths, not the ones we deduced need to be transferred, solely for knowing the total

1.1.90 0.8.0 (Jul 31, 2017) – it is better than ever

A variety of fixes and enhancements

Fixes

- publish would now push merged `git-annex` branch even if no other changes were done
- publish should be able to publish using relative path within SSH URI (git hook would use relative paths)
- publish should better tollerate publishing to pure git and `git-annex` special remotes

Enhancements and new features

- plugin mechanism came to replace export. See export_tarball for the replacement of export. Now it should be easy to extend datalad's interface with custom functionality to be invoked along with other commands.
- Minimalistic coloring of the results rendering
- publish/`copy_to` got progress bar report now and support of `--jobs`
- minor fixes and enhancements to crawler (e.g. support of recursive removes)

1.1.91 0.7.0 (Jun 25, 2017) – when it works - it is quite awesome!

New features, refactorings, and bug fixes.

Major refactoring and deprecations

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

Enhancements and new features

- siblings can now be used to query and configure a local repository by using the sibling name `here`
- siblings can now query and set annex preferred content configuration. This includes `wanted` (as previously supported in other commands), and now also `required`
- New metadata command to interface with datasets/files meta-data
- Documentation for all commands is now built in a uniform fashion
- Significant parts of the documentation of been updated
- Instantiate GitPython's Repo instances lazily

Fixes

- API documentation is now rendered properly as HTML, and is easier to browse by having more compact pages
- Closed files left open on various occasions (Popen PIPEs, etc)
- Restored basic (consumer mode of operation) compatibility with Windows OS

1.1.92 0.6.0 (Jun 14, 2017) – German perfectionism

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

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

Major refactoring and deprecations

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

Fixes

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

Enhancements and new features

- new annotate-paths plumbing command to inspect and annotate provided paths. Use `--modified` to summarize changes between different points in the history
- new clone plumbing command to provide a subset (install a single dataset from a URL) functionality of install
- new diff plumbing command
- new siblings command to list or manipulate siblings
- new subdatasets command to list subdatasets and their properties
- drop and remove commands were refactored
- `benchmarks/` collection of Airspeed velocity benchmarks initiated. See reports at http://datalad.github.io/datalad/
- crawler would try to download a new url multiple times increasing delay between attempts. Helps to resolve problems with extended crawls of Amazon S3
- CRCNS crawler pipeline now also fetches and aggregates meta-data for the datasets from datacite
- overall optimisations to benefit from the aforementioned refactoring and improve user-experience
- a few stub and not (yet) implemented commands (e.g. `move`) were removed from the interface
- Web frontend got proper coloring for the breadcrumbs and some additional caching to speed up interactions. See http://datasets.datalad.org
- Small improvements to the online documentation. See e.g. summary of differences between git/git-annex/datalad

1.1.93 0.5.1 (Mar 25, 2017) – cannot stop the progress

A bugfix release

Fixes

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

Enhancements and new features

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

1.1.94 0.5.0 (Mar 20, 2017) – it's huge

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

Most notable changes

- requires git-annex >= 6.20161210 (or better even >= 6.20161210 for improved functionality)
- commands should now operate on paths specified (if any), without causing side-effects on other dirty/staged files
- save
  - `-a` is deprecated in favor of `-u` or `--all-updates` so only changes known components get saved, and no new files automagically added
  - `-S` does no longer store the originating dataset in its commit message
- add
  - can specify commit/save message with `-m`
- add-sibling and create-sibling
  - now take the name of the sibling (remote) as a `-s` (`--name`) option, not a positional argument
  - `--publish-depends` to setup publishing data and code to multiple repositories (e.g. github + webserve) should now be functional see this comment
  - got `--publish-by-default` to specify what refs should be published by default
  - got `--annex-wanted`, `--annex-groupwanted` and `--annex-group` settings which would be used to instruct annex about preferred content. publish then will publish data using those settings if `wanted` is set.
  - got `--inherit` option to automagically figure out url/wanted and other git/annex settings for new remote sub-dataset to be constructed
- publish
  - got `--skip-failing` refactored into `--missing` option which could use new feature of create-sibling `--inherit`

Fixes

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

Enhancements and new features

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

Internal refactoring

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

1.1.95 0.4.1 (Nov 10, 2016) – CA release

Requires now GitPython >= 2.1.0

Fixes

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

Enhancements and new features

- New rfc822-compliant metadata format
- save
  - -S to save the change also within all super-datasets
- add now has progress-bar reporting
- create-sibling-github to create a :term:`sibling` of a dataset on github

- OpenfMRI crawler and datasets were enriched with URLs to separate files where also available from openfmri s3 bucket (if upgrading your datalad datasets, you might need to run `git annex enableremote datalad` to make them available)
- various enhancements to log messages
- web interface
  - populates "install" box first thus making UX better over slower connections

1.1.96 0.4 (Oct 22, 2016) – Paris is waiting

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

Fixes

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

Enhancements and new features

- interface changes
  - get `--recursion-limit=existing` to not recurse into not-installed subdatasets
  - get `-n` to possibly install sub-datasets without getting any data
  - install `--jobs|-J` to specify number of parallel jobs for annex get call could use (ATM would not work when data comes from archives)
- more (unit-)testing
- documentation: see http://docs.datalad.org/en/latest/basics.html for basic principles and useful shortcuts in referring to datasets
- various webface improvements: breadcrumb paths, instructions how to install dataset, show version from the tags, etc.

1.1.97 0.3.1 (Oct 1, 2016) – what a wonderful week

Primarily bugfixes but also a number of enhancements and core refactorings

Fixes

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

Enhancements and new features

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

1.1.98 0.3 (Sep 23, 2016) – winter is coming

Lots of everything, including but not limited to

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

0.2.3 (Jun 28, 2016) – busy OHBM

New features and bugfix release

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

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

New feature and bugfix release

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

0.2.1 (Jun 10, 2016)

- variety of fixes and enhancements throughout

1.1.99 0.2 (May 20, 2016)

Major RFing to switch from relying on rdf to git native submodules etc

1.1.100 0.1 (Oct 14, 2015)

Release primarily focusing on interface functionality including initial publishing

1.2 Acknowledgments

DataLad development is being performed as part of a US-German collaboration in computational neuroscience (CR-CNS) project "DataGit: converging catalogues, warehouses, and deployment logistics into a federated 'data distribution'" (Halchenko/Hanke), co-funded by the US National Science Foundation (NSF 1429999) and the German Federal Ministry of Education and Research (BMBF 01GQ1411). Additional support is provided by the German federal state of Saxony-Anhalt and the European Regional Development Fund (ERDF), Project: Center for Behavioral Brain Sciences, Imaging Platform

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

1.3 Publications

Further conceptual and technical information on DataLad, and applications built on DataLad, are available from the publications listed below.

The best of both worlds: Using semantic web with JSOB-LD. An example with NIDM Results & DataLad [poster]

- Camille Maumet, Satrajit Ghosh, Yaroslav O. Halchenko, Dorota Jarecka, Nolan Nichols, Jean-Baptist POline, Michael Hanke

One thing to bind them all: A complete raw data structure for auto-generation of BIDS datasets [poster]

- Benjamin Poldrack, Kyle Meyer, Yaroslav O. Halchenko, Michael Hanke

Fantastic containers and how to tame them [poster]

- Yaroslav O. Halchenko, Kyle Meyer, Matt Travers, Dorota Jarecka, Satrajit Ghosh, Jakub Kaczmarzyk, Michael Hanke

YODA: YODA's Organigram on Data Analysis [poster]

- An outline of a simple approach to structuring and conducting data analyses that aims to tightly connect all their essential ingredients: data, code, and computational environments in a transparent, modular, accountable, and practical way.
- Michael Hanke, Kyle A. Meyer, Matteo Visconti di Oleggio Castello, Benjamin Poldrack, Yaroslav O. Halchenko
- F1000Research 2018, 7:1965 (https://doi.org/10.7490/f1000research.1116363.1)

Go FAIR with DataLad [talk]

- On DataLad's capabilities to create and maintain Findable, Accessible, Interoperable, and reusable (FAIR) resources.
- Michael Hanke, Yaroslav O. Halchenko
- Bernstein Conference 2018 workshop: Practical approaches to research data management and reproducibility (slides)
- OpenNeuro kick-off meeting, 2018, Stanford (slide sources)

1.4 Concepts and technologies

1.4.1 Background and motivation

Vision

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

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

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

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

Technological foundation: git-annex

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

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

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

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

Objective

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

1.4.2 Delineation from related solutions

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

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

Data catalogs

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

Data delivery/management middleware

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

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

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

Git/Git-annex/DataLad

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

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

Feature	Git	Git-annex	DataLad
Version control (text, code)	✓	✓can mix	✓can mix
Version control (binary data)	(not advised)	✓	✓
Auto-crawling available resources		✓RSS feeds	✓flexible
Unified dataset handling			✓
• recursive operation on datasets			✓
• seamless operation across datasets boundaries			✓
• meta-data support		✓per-file	✓
• meta-data aggregation			✓flexible
Unified authentication interface			✓

1.4.3 Basic principles

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

Datasets

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

Types of IDs used in datasets

Four types of unique identifiers are used by DataLad to enable identification of different aspects of datasets and their components.

Dataset ID
: A UUID that identifies a dataset as a whole across its entire history and flavors. This ID is stored in a dataset's own configuration file (`<dataset root>/.datalad/config`) under the configuration key `datalad.dataset.id`. As this configuration is stored in a file that is part of the Git history of a dataset, this ID is identical for all "clones" of a dataset and across all its versions. If the purpose or scope of a dataset changes enough to warrant a new dataset ID, it can be changed by altering the dataset configuration setting.

Annex ID
: A UUID assigned to an annex of each individual clone of a dataset repository. Git-annex uses this UUID to track file content availability information. The UUID is available under the configuration key `annex.uuid` and is stored in the configuration file of a local clone (`<dataset root>/.git/config`). A single dataset instance (i.e. clone) can only have a single annex UUID, but a dataset with multiple clones will have multiple annex UUIDs.

Commit ID
: A Git hexsha or tag that identifies a version of a dataset. This ID uniquely identifies the content and history of a dataset up to its present state. As the dataset history also includes the dataset ID, a commit ID of a DataLad dataset is unique to a particular dataset.

Content ID
: Git-annex key (typically a checksum) assigned to the content of a file in a dataset's annex. The checksum reflects the content of a file, not its name. Hence the content of multiple identical files in a single (or across) dataset(s) will have the same checksum. Content IDs are managed by Git-annex in a dedicated `annex` branch of the dataset's Git repository.

Dataset nesting

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

```
~ % datalad create demo
[INFO   ] Creating a new annex repo at /demo/demo
create(ok): /demo/demo (dataset)
~ % cd demo
```

A DataLad dataset is just a Git repo with some initial configuration

```
~/demo % git log --oneline
472e34b (HEAD -> master) [DATALAD] new dataset
f968257 [DATALAD] Set default backend for all files to be MD5E
```

We can generate nested datasets, by telling DataLad to register a new dataset in a parent dataset

```
~/demo % datalad create -d . sub1
[INFO   ] Creating a new annex repo at /demo/demo/sub1
add(ok): sub1 (dataset) [added new subdataset]
add(notneeded): sub1 (dataset) [nothing to add from /demo/demo/sub1]
add(notneeded): .gitmodules (file) [already included in the dataset]
save(ok): /demo/demo (dataset)
create(ok): sub1 (dataset)
action summary:
  add (notneeded: 2, ok: 1)
  create (ok: 1)
  save (ok: 1)
```

A subdataset is nothing more than regular Git submodule

```
~/demo % git submodule
 5f0cddf2026e3fb4864139f27e7415fd72c7d4d0 sub1 (heads/master)
```

Of course subdatasets can be nested

```
~/demo % datalad create -d . sub1/justadir/sub2
[INFO   ] Creating a new annex repo at /demo/demo/sub1/justadir/sub2
add(ok): sub1/justadir/sub2 (dataset) [added new subdataset]
add(notneeded): sub1/justadir/sub2 (dataset) [nothing to add from /demo/demo/sub1/
↪justadir/sub2]
add(notneeded): sub1/.gitmodules (file) [already included in the dataset]
add(notneeded): sub1 (dataset) [already known subdataset]
save(ok): /demo/demo/sub1 (dataset)
save(ok): /demo/demo (dataset)
create(ok): sub1/justadir/sub2 (dataset)
action summary:
  add (notneeded: 3, ok: 1)
  create (ok: 1)
  save (ok: 2)
```

Unlike Git, DataLad automatically takes care of committing all changes associated with the added subdataset up to the given parent dataset

```
~/demo % git status
On branch master
nothing to commit, working tree clean
```

Let's create some content in the deepest subdataset

```
~/demo % mkdir sub1/justadir/sub2/anotherdir
~/demo % touch sub1/justadir/sub2/anotherdir/afile
```

Git can only tell us that something underneath the top-most subdataset was modified

```
~/demo % git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
```

(continues on next page)

(continued from previous page)

```
  (use "git checkout -- <file>..." to discard changes in working directory)
  (commit or discard the untracked or modified content in submodules)

     modified:   sub1 (untracked content)

no changes added to commit (use "git add" and/or "git commit -a")
```

DataLad saves us from further investigation

```
~/demo % datalad diff -r
   modified(dataset): sub1
   modified(dataset): sub1/justadir/sub2
untracked(directory): sub1/justadir/sub2/anotherdir
```

Like Git, it can report individual untracked files, but also across repository boundaries

```
~/demo % datalad diff -r --report-untracked all
   modified(dataset): sub1
   modified(dataset): sub1/justadir/sub2
     untracked(file): sub1/justadir/sub2/anotherdir/afile
```

Adding this new content with Git or git-annex would be an exercise

```
~/demo % git add sub1/justadir/sub2/anotherdir/afile
fatal: Pathspec 'sub1/justadir/sub2/anotherdir/afile' is in submodule 'sub1'
```

DataLad does not require users to determine the correct repository in the tree

```
~/demo % datalad add -d . sub1/justadir/sub2/anotherdir/afile
add(ok): sub1/justadir/sub2/anotherdir/afile (file)
save(ok): /demo/demo/sub1/justadir/sub2 (dataset)
save(ok): /demo/demo/sub1 (dataset)
save(ok): /demo/demo (dataset)
action summary:
  add (ok: 1)
  save (ok: 3)
```

Again, all associated changes in the entire dataset tree, up to the given parent dataset, were committed

```
~/demo % git status
On branch master
nothing to commit, working tree clean
```

DataLad's 'diff' is able to report the changes from these related commits throughout the repository tree

```
~/demo % datalad diff --revision @~1 -r
   modified(dataset): sub1
   modified(dataset): sub1/justadir/sub2
         added(file): sub1/justadir/sub2/anotherdir/afile
```

Dataset collections

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

Benefiting from Git's support for workflows based on decentralized "clones" of a repository, DataLad's datasets can be (re-)published to a new location without losing the connection between the "original" and the new "copy". This is extremely useful for collaborative work, but also in more mundane scenarios such as data backup, or temporary deployment of a dataset on a compute cluster, or in the cloud. Using git-annex, data can also get synchronized across different locations of a dataset (*sibling*s in DataLad terminology). Using metadata tags, it is even possible to configure different levels of desired data redundancy across the network of dataset, or to prevent publication of sensitive data to publicly accessible repositories. Individual datasets in a hierarchy of (sub)datasets need not be stored at the same location. Continuing with an earlier example, it is possible to post a curated collection of datasets, as a superdataset, on GitHub, while the actual datasets live on different servers all around the world.

Basic command line usage

All of DataLad's functionality is available through a single command: *datalad*

Running the datalad command without any arguments, gives a summary of basic options, and a list of available subcommands.

```
~ % datalad
usage: datalad [-h] [-l LEVEL] [-C PATH] [--version]
               [--dbg] [--idbg] [-c KEY=VALUE]
               [-f {default,json,json_pp,tailored,'<template>'}]
               [--report-status {success,failure,ok,notneeded,impossible,error}]
               [--report-type {dataset,file}]
               [--on-failure {ignore,continue,stop}] [--cmd]
               {create,install,get,publish,uninstall,drop,remove,update,create-sibling,
↪create-sibling-github,unlock,save,search,metadata,aggregate-metadata,test,ls,clean,add-
↪archive-content,download-url,run,rerun,addurls,export-archive,extract-metadata,export-
↪to-figshare,no-annex,wtf,add-readme,annotate-paths,clone,create-test-dataset,diff,
↪siblings,sshrun,subdatasets}
               ...
[ERROR  ] Please specify the command
~ % #
```

More comprehensive information is available via the –help long-option (we will truncate the output here)

```
~ % datalad --help       | head -n20
Usage: datalad [global-opts] command [command-opts]

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

*Commands for dataset operations*

  create
      Create a new dataset from scratch
```

(continues on next page)

(continued from previous page)

```
  install
      Install a dataset from a (remote) source
  get
      Get any dataset content (files/directories/subdatasets)
  publish
      Publish a dataset to a known sibling
  uninstall
      Uninstall subdatasets
```

Getting information on any of the available sub commands works in the same way – just pass –help AFTER the subcommand (output again truncated)

```
~ % datalad create --help       | head -n20
Usage: datalad create [-h] [-f] [-D DESCRIPTION] [-d PATH] [--no-annex]
                      [--nosave] [--annex-version ANNEX_VERSION]
                      [--annex-backend ANNEX_BACKEND]
                      [--native-metadata-type LABEL] [--shared-access MODE]
                      [--git-opts STRING] [--annex-opts STRING]
                      [--annex-init-opts STRING] [--text-no-annex]
                      [PATH]

Create a new dataset from scratch.

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

API principles

You can use DataLad's `install` command to download datasets. The command accepts URLs of different protocols (`http`, `ssh`) as an argument. Nevertheless, the easiest way to obtain a first dataset is downloading the default *superdataset* from https://datasets.datalad.org/ using a shortcut.

Downloading DataLad's default superdataset

https://datasets.datalad.org provides a super-dataset consisting of datasets from various portals and sites. Many of them were crawled, and periodically updated, using datalad-crawler extension. The argument `///` can be used as a shortcut that points to the superdataset located at https://datasets.datalad.org/. Here are three common examples in command line notation:

`datalad install ///`
: installs this superdataset (metadata without subdatasets) in a *datasets.datalad.org/* subdirectory under the current directory

`datalad install -r ///openfmri`
: installs the openfmri superdataset into an *openfmri/* subdirectory. Additionally, the `-r` flag recursively downloads

all metadata of datasets available from http://openfmri.org as subdatasets into the *openfmri/* subdirectory

`datalad install -g -J3 -r ///labs/haxby`
installs the superdataset of datasets released by the lab of Dr. James V. Haxby and all subdatasets' metadata. The `-g` flag indicates getting the actual data, too. It does so by using 3 parallel download processes (`-J3` flag).

Downloading datasets via http

In most places where DataLad accepts URLs as arguments these URLs can be regular `http` or `https` protocol URLs. For example:

```
datalad install https://github.com/psychoinformatics-de/studyforrest-data-phase2.git
```

Downloading datasets via ssh

DataLad also supports SSH URLs, such as `ssh://me@localhost/path`.

```
datalad install ssh://me@localhost/path
```

Finally, DataLad supports SSH login style resource identifiers, such as `me@localhost:/path`.

```
datalad install me@localhost:/path
```

Commands *install* vs *get*

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

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

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

So, as a rule of thumb – if you want to install from external URL or fetch a sub-dataset without downloading data files stored under annex – use `install`. In Python API `install` is also to be used when you want to receive in output the corresponding Dataset object to operate on, and be able to use it even if you rerun the script. In all other cases, use `get`.

1.4.4 Credentials

Integration with Git

Git and DataLad can use each other's credential system. Both directions are independent of each other and none is necessarily required. Either direction can be configured based on URL matching patterns. In addition, Git can be configured to always query DataLad for credentials without any URL matching.

Let Git query DataLad

In order to allow Git to query credentials from DataLad, Git needs to be configured to use the git credential helper delivered with DataLad (an executable called *git-credential-datalad*). That is, a section like this needs to be part of one's git config file:

```
[credential "https://*.data.example.com"]
  helper = "datalad"
```

Note:

- This most likely only makes sense at the user or system level (options *–global* `|–system` with *git config*), since cloning of a repository needs the credentials before there is a local repository.
- The name of that section is a URL matching expression - see *man gitcredentials*.
- The URL matching does NOT include the scheme! Hence, if you need to match *http* as well as *https*, you need two such entries.
- Multiple git credential helpers can be configured - Git will ask them one after another until it got a username and a password for the URL in question. For example on macOS, Git comes with a helper to use the system's keychain and Git is configured system-wide to query *git-credential-osxkeychain*. This does not conflict with setting up DataLad's credential helper.
- The example configuration requires *git-credential-datalad* to be in the path in order for Git to find it. Alternatively, the value of the *helper* entry needs to be the absolute path of *git-credential-datalad*.
- In order to make Git always consider DataLad as a credential source, one can simply not specify any URL pattern (so it's *[credential]* instead of *[credential "SOME-PATTERN"]*)

Let DataLad query Git

The other way around, DataLad can ask Git for credentials (which it will acquire via other git credential helpers). To do so, a DataLad provider config needs to be set up:

```
[provider:data_example_provider]
  url_re = https://.*data\.example\.com
  authentication_type = http_basic_auth
  credential = data_example_cred
[credential:data_example_cred]
  type = git
```

Note:

- Such a config lives in a dedicated file named after the provider name (e.g. all of the above example would be the content of `data_example_provider.cfg`, matching *[provider:data_example_provider]*).
- Valid locations for these files are listed in *Credential management*.

- In opposition to Git's approach, *url_re* is a regular expression that matches the entire URL including the scheme.
- The above is particularly important in case of redirects, as DataLad currently matches the URL it was given instead of the one it ultimately uses the credentials with.
- The name of the credential section must match the credential entry in the provider section (e.g. *[credential:data_example_cred]* and *credential = data_example_cred* in the above example).

DataLad will prompt the user to create a provider configuration and respective credentials when it first encounters a URL that requires authentication but no matching credentials are found. This behavior extends to the credential helper and may therefore be triggered by a *git clone* if Git is configured to use *git-credential-datalad*. However, interactivity of *git-credential-datalad* can be turned off (see *git-credential-datalad -h*)

It is possible to end up in a situation where Git would query DataLad and vice versa for the same URL, especially if Git is configured to query DataLad unconditionally. *git-credential-datalad* will discover this circular setup and stop it by simply ignoring DataLad's provider configuration that points back to Git.

1.4.5 Customization and extension of functionality

DataLad provides numerous commands that cover many use cases. However, there will always be a demand for further customization or extensions of built-in functionality at a particular site, or for an individual user. DataLad addresses this need with a mechanism for extending particular DataLad functionality, such as metadata extractor, or providing entire command suites for a specialized purpose.

As the name suggests, a *DataLad extension* package is a proper Python package. Consequently, there is a significant amount of boilerplate code involved in the creation of a new DataLad extension. However, this overhead enables a number of useful features for extension developers:

- extensions can provide any number of additional commands that can be grouped into labeled command suites, and are automatically exposed via the standard DataLad commandline and Python API
- extensions can define *entry_points* for any number of additional metadata extractors that become automatically available to DataLad
- extensions can define *entry_points* for their test suites, such that the standard *datalad create* command will automatically run these tests in addition to the tests shipped with DataLad core
- extensions can ship additional dataset procedures by installing them into a directory `resources/procedures` underneath the extension module directory

Using an extension

A *DataLad extension* is a standard Python package. Beyond installation of the package there is no additional setup required.

Writing your own extensions

A good starting point for implementing a new extension is the "helloworld" demo extension available at https://github.com/datalad/datalad-extension-template. This repository can be cloned and adjusted to suit one's needs. It includes:

- a basic Python package setup
- simple demo command implementation
- Travis test setup

A more complex extension setup can be seen in the DataLad Neuroimaging extension: https://github.com/datalad/datalad-neuroimaging, including additional metadata extractors, test suite registration, and a sphinx-based documentation setup for a DataLad extension.

As a DataLad extension is a standard Python package, an extension should declare dependencies on an appropriate DataLad version, and possibly other extensions via the standard mechanisms.

1.4.6 Design

The chapter described command API principles and the design of particular subsystems in DataLad.

Command line interface

Specification scope and status

This incomplete specification describes the current implementation.

The command line interface (CLI) implementation is located at `datalad.cli`. It provides a console entry point that automatically constructs an `argparse`-based command line parser, which is used to make adequately parameterized calls to the targeted command implementations. It also performs error handling. The CLI automatically supports all commands, regardless of whether they are provided by the core package, or by extensions. It only requires them to be discoverable via the respective extension entry points, and to implement the standard *`datalad.interface.base.Interface`*.

Basic workflow of a command line based command execution

The functionality of the main command line entrypoint described here is implemented in `datalad.cli.main`.

1. Construct an `argparse` parser.
   - this is happening with inspection of the actual command line arguments in order to avoid needless processing
   - when insufficient arguments or other errors are detected, the CLI will fail informatively already at this stage
2. Detect argument completions events, and utilize the parser in a optimized fashion for this purpose.
3. Determine the to-be-executed command from the given command line arguments.
4. Read any configuration overrides from the command line arguments.
5. Change the process working directory, if requested.
6. Execute the target command in one of two modes:
   a. With a basic exception handler
   b. With an exception hook setup that enables dropping into a debugger for any exception that reaches the command line `main()` routine.
7. Unless a debugger is utilized, five error categories are distinguished (in the order given below):
   1. Insufficient arguments (exit code 2)

      A command was called with inadequate or incomplete parameters.

2. Incomplete results (exit code 1)

   While processing an error occurred.

3. A specific internal shell command execution failed (exit code relayed from underlying command)

   The error is reported, as if the command would have been executed directly in the command line. Its output is written to the `stdout`, `stderr` streams, and the exit code of the DataLad process matches the exit code of the underlying command.

4. Keyboard interrupt (exit code 3)

   The process was interrupted by the equivalent of a user hitting `Ctrl+C`.

5. Any other error/exception.

Command parser construction by `Interface` inspection

The parser setup described here is implemented in `datalad.cli.parser`.

A dedicated sub-parser for any relevant DataLad command is constructed. For normal execution use cases, only a single subparser for the target command will be constructed for speed reasons. However, when the command line help system is requested (`--help`) subparsers for all commands (including extensions) are constructed. This can take a considerable amount of time that grows with the number of installed extensions.

The information necessary to configure a subparser for a DataLad command is determined by inspecting the respective *`Interface`* class for that command, and reusing individual components for the parser. This includes:

- the class docstring
- a `_params_` member with a dict of parameter definitions
- a `_examples_` member, with a list of example definitions

All docstrings used for the parser setup will be processed by applying a set of rules to make them more suitable for the command line environment. This includes the processing of `CMD` markup macros, and stripping their `PYTHON` counter parts. Parameter constraint definition descriptions are also altered to exclude Python-specific idioms that have no relevance on the command line (e.g., the specification of `None` as a default).

CLI-based execution of `Interface` command

The execution handler described here is implemented in `datalad.cli.exec`.

Once the main command line entry point determine that a command shall be executed, it triggers a handler function that was assigned and parameterized with the underlying command *`Interface`* during parser construction. At the time of execution, this handler is given the result of `argparse`-based command line argument parsing (i.e., a `Namespace` instance).

From this parser result, the handler constructs positional and keyword arguments for the respective `Interface.__call__()` execution. It does not only process command-specific arguments, but also generic arguments, such as those for result filtering and rendering, which influence the central processing of result recorded yielded by a command.

If an underlying command returns a Python generator it is unwound to trigger the respective underlying processing. The handler performs no error handling. This is left to the main command line entry point.

Provenance capture

Specification scope and status

This specification describes the current implementation.

The ability to capture process provenance—the information what activity initiated by which entity yielded which outputs, given a set of parameters, a computational environment, and potential input data—is a core feature of DataLad.

Provenance capture is supported for any computational process that can be expressed as a command line call. The simplest form of provenance tracking can be implemented by prefixing any such a command line call with `datalad run ...`. When executed in the content of a dataset (with the current working directory typically being in the root of a dataset), DataLad will then:

1. check the dataset for any unsaved modifications
2. execute the given command, when no modifications were found
3. save any changes to the dataset that exist after the command has exited without error

The saved changes are annotated with a structured record that, at minimum, contains the executed command.

This kind of usage is sufficient for building up an annotated history of a dataset, where all relevant modifications are clearly associated with the commands that caused them. By providing more, optional, information to the `run` command, such as a declaration of inputs and outputs, provenance records can be further enriched. This enables additional functionality, such as the automated re-execution of captured processes.

The provenance record

A DataLad provenance record is a key-value mapping comprising the following main items:

- `cmd`: executed command, which may contain placeholders
- `dsid`: DataLad ID of dataset in whose context the command execution took place
- `exit`: numeric exit code of the command
- `inputs`: a list of (relative) file paths for all declared inputs
- `outputs`: a list of (relative) file paths for all declared outputs
- `pwd`: relative path of the working directory for the command execution

A provenance record is stored in a JSON-serialized form in one of two locations:

1. In the body of the commit message created when saving caused the dataset modifications
2. In a sidecar file underneath `.datalad/runinfo` in the root dataset

Sidecar files have a filename (`record_id`) that is based on checksum of the provenance record content, and are stored as LZMA-compressed binary files. When a sidecar file is used, its `record_id` is added to the commit message, instead of the complete record.

Declaration of inputs and outputs

While not strictly required, it is possible and recommended to declare all paths for process inputs and outputs of a command execution via the respective options of `run`.

For all declared inputs, `run` will ensure that their file content is present locally at the required version before executing the command.

For all declared outputs, `run` will ensure that the respective locations are writeable.

It is recommended to declare inputs and outputs both exhaustively and precise, in order to enable the provenance-based automated re-execution of a command. In case of a future re-execution the dataset content may have changed substantially, and a needlessly broad specification of inputs/outputs may lead to undesirable data transfers.

Placeholders in commands and IO specifications

Both command and input/output specification can employ placeholders that will be expanded before command execution. Placeholders use the syntax of the Python `format()` specification. A number of standard placeholders are supported (see the `run` documentation for a complete list):

- `{pwd}` will be replaced with the full path of the current working directory
- `{dspath}` will be replaced with the full path of the dataset that run is invoked on
- `{inputs}` and `{outputs}` expand a space-separated list of the declared input and output paths

Additionally, custom placeholders can be defined as configuration variables under the prefix `datalad.run.substitutions.`. For example, a configuration setting `datalad.run.substitutions.myfile=data.txt` will cause the placeholder `{myfile}` to expand to `data.txt`.

Selection of individual items for placeholders that expand to multiple values is possible via the standard Python `format()` syntax, for example `{inputs[0]}`.

Result records emitted by `run`

When performing a command execution `run` will emit results for:

1. Input preparation (i.e. downloads)
2. Output preparation (i.e. unlocks and removals)
3. Command execution
4. Dataset modification saving (i.e. additions, deletions, modifications)

By default, `run` will stop on the first error. This means that, for example, any failure to download content will prevent command execution. A failing command will prevent saving a potential dataset modification. This behavior can be altered using the standard `on_failure` switch of the `run` command.

The emitted result for the command execution contains the provenance record under the `run_info` key.

Implementation details

Most of the described functionality is implemented by the function `datalad.core.local.run.run_command()`. It is interfaced by the `run` command, but also `rerun`, a utility for automated re-execution based on provenance records, and `containers-run` (provided by the `container` extension package) for command execution in DataLad-tracked containerized environments. This function has a more complex interface, and supports a wider range of use cases than described here.

Application-type vs. library-type usage

Specification scope and status

This specification describes the current implementation.

Historically, DataLad was implemented with the assumption of application-type usage, i.e., a person using DataLad through any of its APIs. Consequently, (error) messaging was primarily targeting humans, and usage advice focused on interactive use. With the increasing utilization of DataLad as an infrastructural component it was necessary to address use cases of library-type or internal usage more explicitly.

DataLad continues to behave like a stand-alone application by default.

For internal use, Python and command-line APIs provide dedicated mode switches.

Library mode can be enabled by setting the boolean configuration setting `datalad.runtime.librarymode` **before the start of the DataLad process**. From the command line, this can be done with the option `-c datalad.runtime.librarymode=yes`, or any other means for setting configuration. In an already running Python process, library mode can be enabled by calling `datalad.enable_libarymode()`. This should be done immediately after importing the `datalad` package for maximum impact.

```
>>> import datalad
>>> datalad.enable_libarymode()
```

In a Python session, library mode **cannot** be enabled reliably by just setting the configuration flag **after** the `datalad` package was already imported. The `enable_librarymode()` function must be used.

Moreover, with `datalad.in_librarymode()` a query utility is provided that can be used throughout the code base for adjusting behavior according to the usage scenario.

Switching back and forth between modes during the runtime of a process is not supported.

A library mode setting is exported into the environment of the Python process. By default, it will be inherited by all child-processes, such as dataset procedure executions.

Library-mode implications

No Python API docs
: Generation of comprehensive doc-strings for all API commands is skipped. This speeds up `import datalad.api` by about 30%.

File URL handling

Specification scope and status

This specification describes the current implementation.

DataLad datasets can record URLs for file content access as metadata. This is a feature provided by git-annex and is available for any annexed file. DataLad improves upon the git-annex functionality in two ways:

1. Support for a variety of (additional) protocols and authentication methods.
2. Support for special URLs pointing to individual files located in registered (annexed) archives, such as tarballs and ZIP files.

These additional features are available to all functionality that is processing URLs, such as `get`, `addurls`, or `download-url`.

Extensible protocol and authentication support

DataLad ships with a dedicated implementation of an external git-annex special remote named `git-annex-remote-datalad`. This is a somewhat atypical special remote, because it cannot receive files and store them, but only supports read operations.

Specifically, it uses the `CLAIMURL` feature of the external special remote protocol to take over processing of URLs with supported protocols in all datasets that have this special remote configured and enabled.

This special remote is automatically configured and enabled in DataLad dataset as a `datalad` remote, by commands that utilize its features, such as `download-url`. Once enabled, DataLad (but also git-annex) is able to act on additional protocols, such as `s3://`, and the respective URLs can be given directly to commands like `git annex addurl`, or `datalad download-url`.

Beyond additional protocol support, the `datalad` special remote also interfaces with DataLad's *Credential management*. It can identify a particular credential required for a given URL (based on something called a "provider" configuration), ask for the credential or retrieve it from a credential store, and supply it to the respective service in an appropriate form. Importantly, this feature neither requires the necessary credential or provider configuration to be encoded in a URL (where it would become part of the git-annex metadata), nor to be committed to a dataset. Hence all information that may depend on which entity is performing a URL request and in what environment is completely separated from the location information on a particular file content. This minimizes the required dataset maintenance effort (when credentials change), and offers a clean separation of identity and availability tracking vs. authentication management.

Indexing and access of archive content

Another git-annex special remote, named `git-annex-remote-datalad-archives`, is used to enable file content retrieval from annexed archive files, such as tarballs and ZIP files. Its implementation concept is closely related to the `git-annex-remote-datalad`, described above. Its main difference is that it claims responsibility for a particular type of "URL" (starting with `dl+archive:`). These URLs encode the identity of an archive file, in terms of its git-annex key name, and a relative path inside this archive pointing to a particular file.

Like `git-annex-remote-datalad`, only read operations are supported. When a request to a `dl+archive:` "URL" is made, the special remote identifies the archive file, if necessary obtains it at the precise version needed, and extracts the respected file content from the archive at the correct location.

This special remote is automatically configured and enabled as `datalad-archives` by the `add-archive-content` command. This command indexes annexed archives, extracts, and registers their content to a dataset. File content availability information is recorded in terms of the `dl+archive:` "URLs", which are put into the git-annex metadata on a file's content.

Result records

Specification scope and status

This specification describes the current implementation.

Result records are the standard return value format for all DataLad commands. Each command invocation yields one or more result records. Result records are routinely inspected throughout the code base, and are used to inform generic error handling, as well as particular calling commands on how to proceed with a specific operation.

The technical implementation of a result record is a Python dictionary. This dictionary must contain a number of mandatory fields/keys (see below). However, an arbitrary number of additional fields may be added to a result record.

The `get_status_dict()` function simplifies the creation of result records.

Note: Developers *must* compose result records with care! DataLad supports custom user-provided hook configurations that use result record fields to decide when to trigger a custom post-result operation. Such custom hooks rely on a persistent naming and composition of result record fields. Changes to result records, including field name changes, field value changes, but also timing/order of record emitting potentially break user set ups!

Mandatory fields

The following keys *must* be present in any result record. If any of these keys is missing, DataLad's behavior is undefined.

`action`

A string label identifying which type of operation a result is associated with. Labels *must not* contain white space. They should be compact, and lower-cases, and use _ (underscore) to separate words in compound labels.

A result without an `action` label will not be processed and is discarded.

`path`

A string with an *absolute* path describing the local entity a result is associated with. Paths must be platform-specific (e.g., Windows paths on Windows, and POSIX paths on other operating systems). When a result is about an entity that has no meaningful relation to the local file system (e.g., a URL to be downloaded), to `path` value should be determined with respect to the potential impact of the result on any local entity (e.g., a URL downloaded to a local file path, a local dataset modified based on remote information).

status

This field indicates the nature of a result in terms of four categories, identified by a string label.

- `ok`: a standard, to-be-expected result
- `notneeded`: an operation that was requested, but found to be unnecessary in order to achieve a desired goal
- `impossible`: a requested operation cannot be performed, possibly because its preconditions are not met
- `error`: an error occurred while performing an operation

Based on the `status` field, a result is categorized into *success* (`ok`, `notneeded`) and *failure* (`impossible`, `error`). Depending on the `on_failure` parameterization of a command call, any failure-result emitted by a command can lead to an `IncompleteResultsError` being raised on command exit, or a non-zero exit code on the command line. With `on_failure='stop'`, an operation is halted on the first failure and the command errors out immediately, with `on_failure='continue'` an operation will continue despite intermediate failures and the command only errors out at the very end, with `on_failure='ignore'` the command will not error even when failures occurred. The latter mode can be used in cases where the initial status-characterization needs to be corrected for the particular context of an operation (e.g., to relabel expected and recoverable errors).

Common optional fields

The following fields are not required, but can be used to enrich a result record with additional information that improves its interpretability, or triggers particular optional functionality in generic result processing.

type

This field indicates the type of entity a result is associated with. This may or may not be the type of the local entity identified by the `path` value. The following values are common, and should be used in matching cases, but arbitrary other values are supported too:

- `dataset`: a DataLad dataset
- `file`: a regular file
- `directory`: a directory
- `symlink`: a symbolic link
- `key`: a git-annex key
- `sibling`: a Dataset sibling or Git remote

message

A message providing additional human-readable information on the nature or provenance of a result. Any non-`ok` results *should* have a message providing information on the rational of their status characterization.

A message can be a string or a tuple. In case of a tuple, the second item can contain values for `%`-expansion of the message string. Expansion is performed only immediately prior to actually outputting the message, hence string formatting runtime costs can be avoided this way, if a message is not actually shown.

logger

If a result record has a `message` field, then a given *Logger* instance (typically from `logging.getLogger()`) will be used to automatically log this message. The log channel/level is determined based on `datalad.log.result-level` configuration setting. By default, this is the `debug` level. When set to `match-status` the log level is determined based on the `status` field of a result record:

- `debug` for `'ok'`, and `'notneeded'` results
- `warning` for `'impossible'` results
- `error` for `'error'` results

This feature should be used with care. Unconditional logging can lead to confusing double-reporting when results rendered and also visibly logged.

refds

This field can identify a path (using the same semantics and requirements as the `path` field) to a reference dataset that represents the larger context of an operation. For example, when recursively processing multiple files across a number of subdatasets, a `refds` value may point to the common superdataset. This value may influence, for example, how paths are rendered in user-output.

parentds

This field can identify a path (using the same semantics and requirements as the `path` field) to a dataset containing an entity.

state

A string label categorizing the state of an entity. Common values are:

- `clean`
- `untracked`
- `modified`
- `deleted`
- `absent`
- `present`

error_message

An error message that was captured or produced while achieving a result.

An error message can be a string or a tuple. In the case of a tuple, the second item can contain values for `%`-expansion of the message string.

`exception`

An exception that occurred while achieving the reported result.

`exception_traceback`

A string with a traceback for the exception reported in `exception`.

Additional fields observed "in the wild"

Given that arbitrary fields are supported in result records, it is impossible to compose a comprehensive list of field names (keys). However, in order to counteract needless proliferation, the following list describes fields that have been observed in implementations. Developers are encouraged to preferably use compatible names from this list, or extend the list for additional items.

In alphabetical order:

`bytesize`
: The size of an entity in bytes (integer).

`gitshasum`
: SHA1 of an entity (string)

`prev_gitshasum`
: SHA1 of a previous state of an entity (string)

`key`
: The git-annex key associated with a `type-file` entity.

`dataset` argument

> **Specification scope and status**
>
> This specification describes the current implementation.

All commands which operate on datasets have a `dataset` argument (`-d` or `--dataset` for the *CLI*) to identify a single dataset as the context of an operation. If `--dataset` argument is not provided, the context of an operation is command-specific. For example, *clone* command will consider the *dataset* which is being cloned to be the context. But typically, a dataset which current working directory belongs to is the context of an operation. In the latter case, if operation (e.g., *get*) does not find a dataset in current working directory, operation fails with an `NoDatasetFound` error.

Impact on relative path resolution

With one exception, the nature of a provided `dataset` argument does **not** impact the interpretation of relative paths. Relative paths are always considered to be relative to the process working directory.

The one exception to this rule is passing a `Dataset` object instance as `dataset` argument value in the Python API. In this, and only this, case, a relative path is interpreted as relative to the root of the respective dataset.

Special values

There are some pre-defined "shortcut" values for dataset arguments:

`^`
: Represents to the topmost superdataset that contains the dataset the current directory is part of.

`^.`
: Represents the root directory of the dataset the current directory is part of.

`///`
: Represents the "default" dataset located under *$HOME/datalad/*.

Use cases

Save modification in superdataset hierarchy

Sometimes it is convenient to work only in the context of a subdataset. Executing a `datalad save <subdataset content>` will record changes to the subdataset, but will leave existing superdatasets dirty, as the subdataset state change will not be saved there. Using the `dataset` argument it is possible to redefine the scope of the save operation. For example:

```
datalad save -d^ <subdataset content>
```

will perform the exact same save operation in the subdataset, but additionally save all subdataset state changes in all superdatasets until the root of a dataset hierarchy. Except for the specification of the dataset scope there is no need to adjust path arguments or change the working directory.

Log levels

Specification scope and status

This specification provides a partial overview of the current implementation.

Log messages are emitted by a wide range of operations within DataLad. They are categorized into distinct levels. While some levels have self-explanatory descriptions (e.g. `warning`, `error`), others are less specific (e.g. `info`, `debug`).

Common principles

Parenthical log message use the same level
: When log messages are used to indicate the start and end of an operation, both start and end message use the same log-level.

Use cases

Command execution

For the `WitlessRunner` and its protocols the following log levels are used:

- High-level execution -> `debug`
- Process start/finish -> `8`
- Threading and IO -> `5`

Drop dataset components

Specification scope and status

This specification is a proposal, subject to review and further discussion. It is now partially implemented in the *drop* command.

§1 The **`drop`** command is the antagonist of **`get`**. Whatever a *drop* can do, should be undoable by a subsequent **`get`** (given unchanged remote availability).

§2 Like **`get`**, **`drop`** primarily operates on a mandatory path specification (to discover relevant files and sudatasets to operate on).

§3 **`drop`** has `--what` parameter that serves as an extensible "mode-switch" to cover all relevant scenarios, like 'drop all file content in the work-tree' (e.g. `--what files`, default, #5858), 'drop all keys from any branch' (i.e. `--what allkeys`, #2328), but also '"drop" AKA uninstall entire subdataset hierarchies' (e.g. `--what all`), or drop preferred content (`--what preferred-content`, #3122).

§4 **`drop`** prevents data loss by default (#4750). Like **`get`** it features a `--reckless` "mode-switch" to disable some or all potentially slow safety mechanism, i.e. 'key available in sufficient number of other remotes', 'main or all branches pushed to remote(s)' (#1142), 'only check availability of keys associated with the worktree, but not other branches'. "Reckless operation" can be automatic, when following a reckless **`get`** (#4744).

§5 **`drop`** properly manages annex lifetime information, e.g. by announcing an annex as `dead` on removal of a repository (#3887).

§6 Like **`get`**, drop supports parallelization #1953

§7 *datalad drop* is not intended to be a comprehensive frontend to *git annex drop* (e.g. limited support for e.g. #1482 outside standard use cases like #2328).

Note: It is understood that the current *uninstall* command is largely or completely made obsolete by this **`drop`** concept.

§8 Given the development in #5842 towards the complete obsolescence of *remove* it becomes necessary to import one of its proposed features:

§9 **`drop`** should be able to recognize a botched attempt to delete a dataset with a plain rm -rf, and act on it in a meaningful way, even if it is just hinting at chmod + rm -rf.

Use cases

The following use cases operate in the dataset hierarchy depicted below:

```
super
├── dir
│   ├── fileD1
│   └── fileD2
├── fileS1
├── fileS2
├── subA
│   ├── fileA
│   ├── subsubC
│   │   ├── fileC
│   └── subsubD
└── subB
    └── fileB
```

Unless explicitly stated, all command are assumed to be executed in the root of *super*.

- U1: `datalad drop fileS1`

  Drops the file content of *file1* (as currently done by **drop**)

- U2: `datalad drop dir`

  Drop all file content in the directory (`fileD{1,2}`; as currently done by **drop**

- U3: `datalad drop subB`

  Drop all file content from the entire *subB* (*fileB*)

- U4: `datalad drop subB --what all`

  Same as above (default `--what files`), because it is not operating in the context of a superdataset (no automatic upward lookups). Possibly hint at next usage pattern).

- U5: `datalad drop -d . subB --what all`

  Drop all from the superdataset under this path. I.e. drop all from the subdataset and drop the subdataset itself (AKA uninstall)

- U6: `datalad drop subA --what all`

  Error: "`subA` contains subdatasets, forgot –recursive?"

- U7: `datalad drop -d . subA -r --what all`

  Drop all content from the subdataset (`fileA`) and its subdatasets (`fileC`), uninstall the subdataset (`subA`) and its subdatasets (`subsubC`, `subsubD`)

- U8: `datalad drop subA -r --what all`

  Same as above, but keep `subA` installed

- U9: `datalad drop sub-A -r`

  Drop all content from the subdataset and its subdatasets (`fileA`, `fileC`)

- U10: `datalad drop . -r --what all`

  Drops all file content and subdatasets, but leaves the superdataset repository behind

- U11: `datalad drop -d . subB`

  Does nothing and hints at alternative usage, see https://github.com/datalad/datalad/issues/5832#issuecomment-889656335

- U12: `cd .. && datalad drop super/dir`

  Like **`get`**, errors because the execution is not associated with a dataset. This avoids complexities, when the given *path*'s point to multiple (disjoint) datasets. It is understood that it could be done, but it is intentionally not done. *datalad -C super drop dir* or *datalad drop -d super super/dir* would work.

Python import statements

Specification scope and status

This specification describes the current (albeit incomplete) implementation.

The following rules apply to any `import` statement in the code base:

- All imports *must* be absolute, unless they import individual pieces of an integrated code component that is only split across several source code files for technical or organizational reasons.
- Imports *must* be placed at the top of a source file, unless there is a specific reason not to do so (e.g., delayed import due to performance concerns, circular dependencies). If such a reason exists, it *must* be documented by a comment at the import statement.
- There *must* be no more than one import per line.
- Multiple individual imports from a single module *must* follow the pattern:

  ```
  from <module> import (
      symbol1,
      symbol2,
  )
  ```

  Individual imported symbols *should* be sorted alphabetically. The last symbol line *should* end with a comma.

- Imports from packages and modules *should* be grouped in categories like
  - Standard library packages
  - 3rd-party packages
  - DataLad core (absolute imports)
  - DataLad extensions
  - DataLad core ("local" relative imports)

  Sorting imports can be aided by https://github.com/PyCQA/isort (e.g. `python -m isort -m3 --fgw 2 --tc <filename>`).

Examples

```
   from collections import OrderedDict
   import logging
   import os

   from datalad.utils import (
       bytes2human,
       ensure_list,
       ensure_unicode,
       get_dataset_root as gdr,
   )

In the `datalad/submodule/tests/test_mod.py` test file demonstrating an "exception" to␣
↪absolute imports
rule where test files are accompanying corresponding files of the underlying module::

   import os

   from datalad.utils import ensure_list

   from ..mod import func1

   from datalad.tests.utils_pytest import assert_true
```

Miscellaneous patterns

DataLad is the result of a distributed and collaborative development effort over many years. During this time the scope of the project has changed multiple times. As a consequence, the API and employed technologies have been adjusted repeatedly. Depending on the age of a piece of code, a clear software design is not always immediately visible. This section documents a few design patterns that the project strives to adopt at present. Changes to existing code and new contributions should follow these guidelines.

Generator methods in *Repo* classes

Substantial parts of DataLad are implemented to behave like Python generators in order to be maximally responsive when processing long-running tasks. This included methods of the core API classes `GitRepo` and `AnnexRepo`. By convention, such methods carry a trailing _ in their name. In some cases, sibling methods with the same name, but without the trailing underscore are provided. These behave like their generator-equivalent, but eventually return an iterable once processing is fully completed.

Calls to Git commands

DataLad is built on Git, so calls to Git commands are a key element of the code base. All such calls should be made through methods of the `GitRepo` class. This is necessary, as only there it is made sure that Git operates under the desired conditions (environment configuration, etc.).

For some functionality, for example querying and manipulating *gitattributes*, dedicated methods are provided. However, in many cases simple one-off calls to get specific information from Git, or trigger certain operations are needed. For these purposes the `GitRepo` class provides a set of convenience methods aiming to cover use cases requiring particular return values:

- test success of a command: `call_git_success()`
- obtain *stdout* of a command: `call_git()`
- obtain a single output line: `call_git_oneline()`
- obtain items from output split by a separator: `call_git_items_()`

All these methods take care of raising appropriate exceptions when expected conditions are not met. Whenever desired functionality can be achieved using simple custom calls to Git via these methods, their use is preferred over the implementation of additional, dedicated wrapper methods.

Command examples

Examples of Python and commandline invocations of DataLad's user-oriented commands are defined in the class of the respective command as dictionaries within *_examples_*:

```
_examples_ = [
 dict(text="""Create a dataset 'mydataset' in the current directory""",
      code_py="create(path='mydataset')",
      code_cmd="datalad create mydataset",
 dict(text="""Apply the text2git procedure upon creation of a dataset""",
      code_py="create(path='mydataset', cfg_proc='text2git')",
      code_cmd="datalad create -c text2git mydataset")
      ]
```

The formatting of code lines is preserved. Changes to existing examples and new contributions should provide examples for Python and commandline API, as well as a concise description.

Exception handling

Specification scope and status

This specification describes the current implementation target.

Catching exceptions

Whenever we catch an exception in an `except` clause, the following rules apply:

- unless we (re-)raise, the first line instantiates a `CapturedException`:

  ```
  except Exception as e:
      ce = CapturedException(e)
  ```

  First, this ensures a low-level (8) log entry including the traceback of that exception. The depth of the included traceback can be limited by setting the `datalad.exc.str.tb_limit` config accordingly.

  Second, it deletes the frame stack references of the exception and keeps textual information only, in order to avoid circular references, where an object (whose method raised the exception) isn't going to be picked by the garbage collection. This can be particularly troublesome if that object holds a reference to a subprocess for example. However, it's not easy to see in what situation this would really be needed and we never need anything other than the textual information about what happened. Making the reference cleaning a general rule is easiest to write, maintain and review.

- if we raise, neither a log entry nor such a `CapturedException` instance is to be created. Eventually, there will be a spot where that (re-)raised exception is caught. This then is the right place to log it. That log entry will have the traceback, there's no need to leave a trace by means of log messages!

- if we raise, but do not simply reraise that exact same exception, in order to change the exception class and/or its message, `raise from` must be used!:

  ```
  except SomeError as e:
      raise NewError("new message") from e
  ```

  This ensures that the original exception is properly registered as the cause for the exception via its `__cause__` attribute. Hence, the original exception's traceback will be part of the later on logged traceback of the new exception.

Messaging about an exception

In addition to the auto-generated low-level log entry there might be a need to create a higher-level log, a user message or a (result) dictionary that includes information from that exception. While such messaging may use anything the (captured) exception provides, please consider that "technical" details about an exception are already auto-logged and generally not incredibly meaningful for users.

For message creation `CapturedException` comes with a couple of `format_*` helper methods, its `__str__` provides a short representation of the form `ExceptionClass(message)` and its `__repr__` the log form with a traceback that is used for the auto-generated log.

For result dictionaries `CapturedException` can be assigned to the field `exception`. Currently, `get_status_dict` will consider this field and create an additional field with a traceback string. Hence, whether putting a captured exception into that field actually has an effect depends on whether `get_status_dict` is subsequently used with that dictionary. In the future such functionality may move into result renderers instead, leaving the decision of what to do with the passed `CapturedException` to them. Therefore, even if of no immediate effect, enhancing the result dicts accordingly makes sense already, since it may be useful when using datalad via its python interface already and provide instant benefits whenever the result rendering gets such an upgrade.

Credential management

Specification scope and status

This specification describes the current implementation.

Various components of DataLad need to be passed credentials to interact with services that require authentication. This includes downloading files, but also things like REST API usage or authenticated cloning. Key components of DataLad's credential management are credentials types, providers, authenticators and downloaders.

Credentials

Supported credential types include basic user/password combinations, access tokens, and a range of tailored solutions for particular services. All credential type implementations are derived from a common `Credential` base class. A mapping from string labels to credential classes is defined in `datalad.downloaders.CREDENTIAL_TYPES`.

Importantly, credentials must be identified by a name. This name is a label that is often hard-coded in the program code of DataLad, any of its extensions, or specified in a dataset or in provider configurations (see below).

Given a credential `name`, one or more credential `component`(s) (e.g., `token`, `username`, or `password`) can be looked up by DataLad in at least two different locations. These locations are tried in the following order, and the first successful lookup yields the final value.

1. A configuration item `datalad.credential.<name>.<component>`. Such configuration items can be defined in any location supported by DataLad's configuration system. As with any other specification of configuration items, environment variables can be used to set or override credentials. Variable names take the form of `DATALAD_CREDENTIAL_<NAME>_<COMPONENT>`, and standard replacement rules into configuration variable names apply.
2. DataLad uses the *keyring* package https://pypi.org/project/keyring to connect to any of its supported back-ends for setting or getting credentials, via a wrapper in `keyring_`. This provides support for credential storage on all major platforms, but also extensibility, providing 3rd-parties to implement and use specialized solutions.

When a credential is required for operation, but could not be obtained via any of the above approaches, DataLad can prompt for credentials in interactive terminal sessions. Interactively entered credentials will be stored in the active credential store available via the `keyring` package. Note, however, that the keyring approach is somewhat abused by datalad. The wrapper only uses `get_/set_password` of `keyring` with the credential's `FIELDS` as the name to query (essentially turning the keyring into a plain key-value store) and "datalad-<CREDENTIAL-LABEL>" as the "service name". With this approach it's not possible to use credentials in a system's keyring that were defined by other, datalad unaware software (or users).

When a credential value is known but invalid, the invalid value must be removed or replaced in the active credential store. By setting the configuration flag `datalad.credentials.force-ask`, DataLad can be instructed to force interactive credential re-entry to effectively override any store credential with a new value.

Providers

Providers are associating credentials with a context for using them and are defined by configuration files. A single provider is represented by `Provider` object and the list of available providers is represented by the `Providers` class. A provider is identified by a label and stored in a dedicated config file per provider named *LABEL.cfg*. Such a file can reside in a dataset (under *.datalad/providers/*), at the user level (under *{user_config_dir}/providers*), at the system level (under *{site_config_dir}/providers*) or come packaged with the datalad distribution (in directory *configs* next to *providers.py*). Such a provider specifies a regular expression to match URLs against and assigns authenticator abd credentials to be used for a match. Credentials are referenced by their label, which in turn is the name of another section in such a file specifying the type of the credential. References to credential and authenticator types are strings that are mapped to classes by the following dict definitions:

- `datalad.downloaders.AUTHENTICATION_TYPES`
- `datalad.downloaders.CREDENTIAL_TYPES`

Available providers can be loaded by `Providers.from_config_files` and `Providers.get_provider(url)` will match a given URL against them and return the appropriate *Provider* instance. A `Provider` object will determine a downloader to use (derived from `BaseDownloader`), based on the URL's protocol.

Note, that the provider config files are not currently following datalad's general config approach. Instead they are special config files, read by `configparser.ConfigParser` that are not compatible with *git-config* and hence the `ConfigManager`.

There are currently two ways of storing a provider and thus creating its config file: `Providers.enter_new` and `Providers._store_new`. The former will only work interactively and provide the user with options to choose from, while the latter is non-interactive and can therefore only be used, when all properties of the provider config are known and passed to it. There's no way at the moment to store an existing `Provider` object directly.

Integration with Git

In addition, there's a special case for interfacing *git-credential*: A dedicated `GitCredential` class is used to talk to Git's `git-credential` command instead of the keyring wrapper. This class has identical fields to the `UserPassword` class and thus can be used by the same authenticators. Since Git's way to deal with credentials doesn't involve labels but only matching URLs, it is - in some sense - the equivalent of datalad's provider layer. However, providers don't talk to a backend, credentials do. Hence, a more seamless integration requires some changes in the design of datalad's credential system as a whole.

In the opposite direction - making Git aware of datalad's credentials, there's no special casing, though. DataLad comes with a *git-credential-datalad* executable. Whenever Git is configured to use it by setting *credential.helper=datalad*, it will be able to query datalad's credential system for a provider matching the URL in question and retrieve the referenced by this provider credentials. This helper can also store a new provider+credentials when asked to do so by Git. It can do this interactively, asking a user to confirm/change that config or - if *credential.helper='datalad –non-interactive'* - try to non-interactively store with its defaults.

Authenticators

Authenticators are used by downloaders to issue authenticated requests. They are not easily available to directly be applied to requests being made outside of the downloaders.

URL substitution

Specification scope and status

This specification describes the current implementation. This implementation is covering URL substitution in `clone` only. A further extension to URL processing elsewhere is possible.

URL substitution is a transformation of a given URL using a set of specifications. Such specification can be provided as configuration settings (via all supported configuration sources). These configuration items must follow the naming scheme `datalad.clone.url-substitute.<label>`, where `<label>` is an arbitrary identifier.

A substitution specification is a string with a match and substitution expression, each following Python's regular expression syntax. Both expressions are concatenated into a single string with an arbitrary delimiter character. The delimiter is defined by prefixing the string with the delimiter. Prefix and delimiter are stripped from the expressions before processing. Example:

```
,^http://(.*)$,https://\\1
```

A particular configuration item can be defined multiple times (see examples below) to form a substitution series. Substitutions in the same series will be applied incrementally, in order of their definition. If the first substitution expression does not match, the entire series will be ignored. However, following a first positive match all further substitutions in a series are processed, regardless whether intermediate expressions match or not.

Any number of substitution series can be configured. They will be considered in no particular order. Consequently, it advisable to implement the first match specification of any series as specific as possible, in order to prevent undesired transformations.

Examples

Change the protocol component of a given URL in order to hand over further processing to a dedicated Git remote helper. Specifically, the following example converts Open Science Framework project URLs like `https://osf.io/f5j3e/` into `osf://f5j3e`, a URL that can be handle by `git-remote-osf`, the Git remote helper provided by the datalad-osf extension package:

```
datalad.clone.url-substitute.osf = ,^https://osf.io/([^/]+)[/]*$,osf://\1
```

Here is a more complex examples with a series of substitutions. The first expression ensures that only GitHub URLs are being processed. The associated substitution disassembles the URL into its two only relevant components, the organisation/user name, and the project name:

```
datalad.clone.url-substitute.github = ,https?://github.com/([^/]+)/(.*)$,\1###\2
```

All other expressions in this series that are described below will only be considered if the above expression matched.

The next two expressions in the series normalize URL components that maybe be auto-generated by some DataLad functionality, e.g. subdataset location candidate generation from directory names:

```
# replace (back)slashes with a single dash
datalad.clone.url-substitute.github = ,[/\\]+,-

# replace with whitespace (URL-quoted or not) with a single underscore
datalad.clone.url-substitute.github = ,\s+|(%2520)+|(%20)+,_
```

The final expression in the series is recombining the organization/user name and project name components back into a complete URL:

```
datalad.clone.url-substitute.github = ,([^#]+)###(.*),https://github.com/\1/\2
```

Threaded runner

Specification scope and status

This specification provides an overview over the current implementation of the subprocess runner that is used throughout datalad.

Threads

DataLad often requires the execution of subprocesses. While subprocesses are executed, datalad, i.e. its main thread, should be able to read data from stdout and stderr of the subprocess as well as write data to stdin of the subprocess. This requires a way to efficiently multiplex reading from stdout and stderr of the subprocess as well as writing to stdin of the subprocess.

Since non-blocking IO and waiting on multiple sources (poll or select) differs vastly in terms of capabilities and API on different OSs, we decided to use blocking IO and threads to multiplex reading from different sources.

Generally we have a number of threads that might be created and executed, depending on the need for writing to stdin or reading from stdout or stderr. Each thread can read from either a single queue or a file descriptor. Reading is done blocking. Each thread can put data into multiple queues. This is used to transport data that was read as well as for signaling conditions like closed file descriptors.

Conceptually, there are the main thread and two different types of threads:

- type 1: transport threads (1 thread per process I/O descriptor)
- type 2: process waiting thread (1 thread)

Transport Threads

Besides the main thread, there might be up to three additional threads to handle data transfer to `stdin`, and from `stdout` and `stderr`. Each of those threads copies data between queues and file descriptors in a tight loop. The stdin-thread reads from an input-queue, the stdout- and stderr-threads write to an output queue. Each thread signals its exit to a set of signal queues, which might be identical to the output queues.

The `stdin`-thread reads data from a queue and writes it to the `stdin`-file descriptor of the sub-process. If it reads `None` from the queue, it will exit. The thread will also exit, if an exit is requested by calling `thread.request_exit()`, or if an error occurs during writing. In all cases it will enqueue a `None` to all its signal-queues.

The stdout- and stderr-threads read from the respective file descriptor and enqueue data into their output queue, unless the data has zero length (which indicates a closed descriptor). On a zero-length read they exit and enqueue None into their signal queues.

All queues are infinite. Nevertheless signaling is performed with a timeout of one 100 milliseconds in order to ensure that threads can exit.

Process Waiting Thread

The process waiting thread waits for a given process to exit and enqueues an exit notification into it signal queues.

Main Thread

There is a single queue, the output_queue, on which the main thread waits, after all transport threads, and the process waiting thread are started. The output_queue is the signaling queue and the output queue of the stderr-thread and the stdout-thread. It is also the signaling queue of the stdin-thread, and it is the signaling queue for the process waiting threads.

The main thread waits on the output_queue for data or signals and handles them accordingly, i.e. calls data callbacks of the protocol if data arrives, and calls connection-related callbacks of the protocol if other signals arrive. If no messages arrive on the output_queue, the main thread blocks for 100ms. If it is unblocked, either by getting a message or due to elapsing of the 100ms, it will process timeouts. If the timeout-parameter to the constructor was not None, it will check the last time any of the monitored files (stdout and/or stderr) yielded data. If the time is larger than the specified timeout, it will call the timeout method of the protocol instance. Due to this implementation, the resolution for timeouts is 100ms. The main thread handles the closing of stdin-, stdout-, and stderr-file descriptors if all other threads have terminated and if output_queue is empty. These tasks are either performed in the method ThreadedRunner.run() or in a result generator that is returned by ThreadedRunner.run() whenever send() is called on it.

Protocols

Due to its history datalad uses the protocol defined in asyncio.protocols.SubprocessProtocol and in asyncio.protocols.BaseProtocol. To keep compatibility with the code base, the threaded-runner implementation uses the same interface. Please note, although we use the same interface and although the interface is defined in the asyncio libraries, the threaded-runner implementation does not make any use of asyncio. The description of the interface nevertheless applies in the context of the threaded-runner. The following methods of the SubprocessProtocol are supported.

- SubprocessProtocol.pipe_data_received(fd, data)
- SubprocessProtocol.pipe_connection_lost(fd, exc)
- SubprocessProtocol.process_exited()

In addition the following methods of BaseProtocol are supported:

- BaseProtocol.connection_made(transport)
- BaseProtocol.connection_lost(exc)

The datalad-provided protocol datalad.runners.protocol.WitlessProtocol provides an additional callback:

- WitlessProtocol.timeout(fd)

The method timeout() will be called when the parameter timeout in WitlessRunner.run, ThreadedRunner.run, or run_command is set to a number specifying the desired timeout in seconds. If no data is received from stdin,

or `stderr` (if those are supposed to be captured), the method `WitlessProtocol.timeout(fd)` is called with `fd` set to the respective file number, e.g. 1, or 2. If `WitlessProtocol.timeout(fd)` returns `True`, only the corresponding file descriptor will be closed and the associated threads will exit.

The method `WitlessProtocol.timeout(fd)` is also called if stdout, stderr and stdin are closed and the process does not exit within the given interval. In this case `fd` is set to `None`. If `WitlessProtocol.timeout(fd)` returns `True` the process is terminated.

Object and Generator Results

If the protocol that is provided to `run()` does not inherit `datalad.runner.protocol.GeneratorMixIn`, the final result that will be returned to the caller is determined by calling `WitlessProtocol._prepare_result()`. Whatever object this method returns will be returned to the caller.

If the protocol that is provided to `run()` does inherit `datalad.runner.protocol.GeneratorMixIn`, `run()` will return a `Generator`. This generator will yield the elements that were sent to it in the protocol-implementation by calling `GeneratorMixIn.send_result()` in the order in which the method `GeneratorMixIn.send_result()` is called. For example, if `GeneratorMixIn.send_result(43)` is called, the generator will yield `43`, and if `GeneratorMixIn.send_result({"a": 123, "b": "some data"})` is called, the generator will yield `{"a": 123, "b": "some data"}`.

Internally the generator is implemented by keeping track of the process state and waiting in the `output_queue` once, when `send` (or `__next__`) is called on it.

BatchedCommand and BatchedAnnex

Specification scope and status

This specification describes the new implementation of `BatchedCommand` and `BatchedAnnex` in `datalad`.

Batched Command

The class `BatchedCommand` (in `datalad.cmd`), holds an instance of a running subprocess, allows to send requests to the subprocess over its stdin, and to receive responses from the subprocess over its stdout.

Requests can be provided to an instance of `BatchedCommand` by passing a single request or a list of requests to `BatchCommand.__call__()`, i.e. by applying the function call-operator to an instance of `BatchedCommand`. A request is either a string or a tuple of strings. In the latter case, the elements of the tuple will be joined by `" "`. More than one request can be given by providing a list of requests, i.e. a list of strings or tuples. In this case, the return value will be a list with one response for every request.

`BatchedCommand` will send each request that is sent to the subprocess as a single line, after terminating the line by `"\n"`. After the request is sent, `BatchedCommand` calls an output-handler with stdout-ish (an object that provides a `readline()`-function which operates on the stdout of the subprocess) of the subprocess as argument. The output-handler can be provided to the constructor. If no output-handler is provided, a default output-handler is used. The default output-handler reads a single output line on stdout, using `io.IOBase.readline()`, and returns the `rstrip()`-ed line.

The subprocess must at least emit one line of output per line of input in order to prevent the calling thread from blocking. In addition, the size of the output, i.e. the number of lines that the result consists of, must be discernible by the output-handler. That means, the subprocess must either return a fixed number of lines per input line, or it must indicate the end of a result in some other way, e.g. with an empty line.

Remark: In principle any output processing could be performed. But, if the output-handler blocks on stdout, the calling thread will be blocked. Due to the limited capabilities of the stdout-ish that is passed to the output-handler, the output-handler must rely on `readline()` to process the output of the subprocess. Together with the line-based request sending, `BatchedCommand` is geared towards supporting the batch processing modes of `git` and `git-annex`. *This has to be taken into account when providing a custom output handler.*

When `BatchedCommand.close()` is called, stdin, stdout, and stderr of the subprocess are closed. This indicates the end of processing to the subprocess. Generally the subprocess is expected to exit shortly after that. `BatchedCommand.close()` will wait for the subprocess to end, if the configuration `datalad.runtime.stalled-external` is set to `"wait"`. If the configuration `datalad.runtime.stalled-external` is set to `"abandon"`, `BatchedCommand.close()` will return after "timeout" seconds if `timeout` was provided to `BatchedCommand.__init__()`, otherwise it will return after 11 seconds. If a timeout occurred, the attribute `wait_timed_out` of the `BatchedCommand` instance will be set to `True`. If `exception_on_timeout=True` is provided to `BatchedCommand.__init__()`, a `subprocess.TimeoutExpired` exception will be raised on a timeout while waiting for the process. It is not safe to reused a `BatchedCommand` instance after such an exception was risen.

Stderr of the subprocess is gathered in a byte-string. Its content will be returned by `BatchCommand.close()` if the parameter `return_stderr` is `True`.

Implementation details

`BatchedCommand` uses `WitlessRunner` with a protocol that has `datalad.runner.protocol.GeneratorMixIn` as a super-class. The protocol uses an output-handler to process data, if an output-handler was specified during construction of `BatchedCommand`.

`BatchedCommand.close()` queries the configuration key `datalad.runtime.stalled-external` to determine how to handle non-exiting processes (there is no killing, processes or process zombies might just linger around until the next reboot).

The current implementation of `BatchedCommand` can process a list of multiple requests at once, but it will collect all answers before returning a result. That means, if you send 1000 requests, `BatchedCommand` will return after having received 1000 responses.

BatchedAnnex

`BatchedAnnex` is a subclass of `BatchedCommand` (which it actually doesn't have to be, it just adds git-annex specific parameters to the command and sets a specific output handler).

`BatchedAnnex` provides a new output-handler if the constructor-argument `json` is `True`. In this case, an output handler is used that reads a single line from stdout, strips the line and converts it into a json object, which is returned. If the stripped line is empty, an empty dictionary is returned.

Standard parameters

Specification scope and status

This specification partially describes the current implementation, and partially is a proposal, subject to review and further discussion.

Several "standard parameters" are used in various DataLad commands. Those standard parameters have an identical meaning across the commands they are used in. Commands should ensure that they use those "standard parameters" where applicable and do not deviate from the common names nor the common meaning.

Currently used standard parameters are listed below, as well as suggestions on how to harmonize currently deviating standard parameters. Deviations from the agreed upon list should be harmonized. The parameters are listed in their command-line form, but similar names and descriptions apply to their Python form.

-d/--dataset
A pointer to the dataset that a given command should operate on

--dry-run
Display details about the command execution without actually running the command.

-f/--force
Enforce the execution of a command, even when certain security checks would normally prevent this

-J/--jobs
Number of parallel jobs to use.

-m/--message
A commit message to attach to the saved change of a command execution.

-r/--recursive
Perform an operation recursively across subdatasets

-R/--recursion-limit
Limit recursion to a given amount of subdataset levels

-s/--sibling-name [SUGGESTION]
The identifier for a dataset sibling (remote)

Certain standard parameters will have their own design document. Please refer to those documents for more in-depth information.

Positional vs Keyword parameters

Specification scope and status

This specification is a proposal, subject to review and further discussion. Technical preview was implemented in the PR #6176.

Motivation

Python allows for keyword arguments (arguments with default values) to be specified positionally. That complicates addition or removal of new keyword arguments since such changes must account for their possible positional use. Moreover, in case of our Interface's, it contributes to inhomogeneity since when used in *CLI*, all keyword arguments must be specified via non-positional `--<option>`'s, whenever Python interface allows for them to be used positionally.

Python 3 added possibility to use a * separator in the function definition to mandate that all keyword arguments *after* it must be be used only via keyword (`<option>=<value>`) specification. It is encouraged to use * to explicitly separate out positional from keyword arguments in majority of the cases, and below we outline two major types of constructs.

Interfaces

Subclasses of the *Interface* provide specification and implementation for both *CLI* and Python API interfaces. All new interfaces must separate all CLI `--options` from positional arguments using `*` in their `__call__` signature.

Note: that some positional arguments could still be optional (e.g., destination `path` for `clone`), and thus should be listed **before** `*`, despite been defined as a keyword argument in the `__call__` signature.

A unit-test will be provided to guarantee such consistency between *CLI* and Python interfaces. Overall, exceptions to this rule could be only some old(er) interfaces.

Regular functions and methods

Use of `*` is encouraged for any function (or method) with keyword arguments. Generally, `*` should come before the first keyword argument, but similarly to the Interfaces above, it is left to the discretion of the developer to possibly allocate some (just few) arguments which could be used positionally if specified.

Docstrings

Specification scope and status

This specification provides a partial overview of the current implementation.

Docstrings in DataLad source code are used and consumed in many ways. Besides serving as documentation directly in the sources, they are also transformed and rendered in various ways.

- Command line `--help` output
- Python's `help()` or IPython's `?`
- Manpages
- Sphinx-rendered documentation for the Python API and the command line API

A common source docstring is transformed, amended and tuned specifically for each consumption scenario.

Formatting overview and guidelines

In general, the docstring format follows the NumPy standard. In addition, we follow the guidelines of Restructured Text with the additional features and treatments provided by Sphinx, and some custom formatting outlined below.

Version information

Additions, changes, or deprecation should be recorded in a docstring using the standard Sphinx directives `versionadded`, `versionchanged`, `deprecated`:

```
.. deprecated:: 0.16
   The ``dryrun||--dryrun`` option will be removed in a future release, use
   the renamed ``dry_run||--dry-run`` option instead.
```

API-conditional docs

The CMD and PY macros can be used to selectively include documentation for specific APIs only:

```
options to pass to :command:`git init`. [PY: Options can be given as a list
of command line arguments or as a GitPython-style option dictionary PY][CMD:
Any argument specified after the destination path of the repository will be
passed to git-init as-is CMD].
```

For API-alternative command and argument specifications the following format can be used:

```
``<python-api>||<cmdline-api``
```

where the double backticks are mandatory and <python-part> and <cmdline-part> represent the respective argument specification for each API. In these specifications only valid argument/command names are allowed, plus a comma character to list multiples, and the dot character to include an ellipsis:

```
``github_organization||-g,--github-organization``

``create_sibling_...||create-sibling-...``
```

Reflow text

When automatic transformations negatively affect the presentation of a docstring due to excessive removal of content, leaving "holes", the REFLOW macro can be used to enclose such segments, in order to reformat them as the final processing step. Example:

```
|| REFLOW >>
The API has been aligned with the some
``create_sibling_...||create-sibling-...`` commands of other GitHub-like
services, such as GOGS, GIN, GitTea.<< REFLOW ||
```

The start macro must appear on a dedicated line.

Progress reporting

Specification scope and status

This specification describes the current implementation.

Progress reporting is implemented via the logging system. A dedicated function *datalad.log.log_progress()* represents the main API for progress reporting. For some standard use cases, the utilities *datalad.log.with_progress()* and *datalad.log.with_result_progress()* can simplify result reporting further.

Design and implementation

This basic idea is to use an instance of datalad's loggers to emit log messages with particular attributes that are picked up by `datalad.log.ProgressHandler` (derived from `logging.Handler`), and are acted on differently, depending on configuration and conditions of a session (e.g., interactive terminal sessions vs. non-interactive usage in scripts). This variable behavior is implemented via the use of `logging` standard library log filters and handlers. Roughly speaking, `datalad.log.ProgressHandler` will only be used for interactive sessions. In non-interactive cases, progress log messages are inspected by *datalad.log.filter_noninteractive_progress()*, and are either discarded or treated like any other log message (see *datalad.log.LoggerHelper.get_initialized_logger()* for details on the handler and filter setup).

`datalad.log.ProgressHandler` inspects incoming log records for attributes with names starting with *dlm_progress*. It will only process such records and pass others on to the underlying original log handler otherwise.

`datalad.log.ProgressHandler` takes care of creating, updating and destroying any number of simultaneously running progress bars. Progress reports must identify the respective process via an arbitrary string ID. It is the caller's responsibility to ensure that this ID is unique to the target process/activity.

Reporting progress with *log_progress()*

Typical progress reporting via *datalad.log.log_progress()* involves three types of calls.

1. Start reporting progress about a process

A typical call to start of progress reporting looks like this

```
log_progress(
    # the callable used to emit log messages
    lgr.info,
    # a unique identifiers of the activity progress is reported for
    identifier,
    # main message
    'Unlocking files',
    # optional unit string for a progress bar
    unit=' Files',
    # optional label to be displayed in a progress bar
    label='Unlocking',
    # maximum value for a progress bar
    total=nfiles,
)
```

A new progress bar will be created automatically for any report with a previously unseen activity `identifier`. It can be configured via the specification of a number of arguments, most notably a target `total` for the progress bar. See *datalad.log.log_progress()* for a complete overview.

Starting a progress report must be done with a dedicated call. It cannot be combined with a progress update.

2. Update progress information about a process

Any subsequent call to *datalad.log.log_progress()* with an activity identifier that has already been seen either updates, or finishes the progress reporting for an activity. Updates must contain an `update` key which either specifies a new value (if *increment=False*, the default) or an increment to previously known value (if *increment=True*):

```
log_progress(
    lgr.info,
    # must match the identifier used to start the progress reporting
    identifier,
    # arbitrary message content, string expansion supported just like
    # regular log messages
    "Files to unlock %i", nfiles,
    # critical key for report updates
    update=1,
    # ``update`` could be an absolute value or an increment
    increment=True
)
```

Updating a progress report can only be done after a progress reporting was initialized (see above).

3. Report completion of a process

A progress bar will remain active until it is explicitly taken down, even if an initially declared `total` value may have been reached. Finishing a progress report requires a final log message with the corresponding identifiers which, like the first initializing message, does NOT contain an `update` key.

```
log_progress(
    lgr.info,
    identifier,
    # closing log message
    "Completed unlocking files",
)
```

Progress reporting in non-interactive sessions

datalad.log.log_progress() takes a *noninteractive_level* argument that can be used to specify a log level at which progress is logged when no progress bars can be used, but actual log messages are produced.

```
import logging

log_progress(
    lgr.info,
    identifier,
    "Completed unlocking files",
    noninteractive_level=logging.INFO
)
```

Each call to *log_progress()* can be given a different log level, in order to control the verbosity of the reporting in such a scenario. For example, it is possible to log the start or end of an activity at a higher level than intermediate updates. It is also possible to single out particular intermediate events, and report them at a higher level.

If no *noninteractive_level* is specified, the progress update is unconditionally logged at the level implied by the given logger callable.

Reporting progress with *with_(result_)progress()*

For cases were a list of items needs to be processes sequentially, and progress shall be communicated, two additional helpers could be used: the decorators `datalad.log.with_progress()` and `datalad.log.with_result_progress()`. They require a callable that takes a list (or more generally a sequence) of items to be processed as the first positional argument. They both set up and perform all necessary calls to `log_progress()`.

The difference between these helpers is that `datalad.log.with_result_progress()` expects a callable to produce DataLad result records, and supports customs filters to decide which particular result records to consider for progress reporting (e.g., only records for a particular *action* and *type*).

Output non-progress information without interfering with progress bars

`log_progress()` can also be useful when not reporting progress, but ensuring that no other output is interfering with progress bars, and vice versa. The argument *maint* can be used in this case, with no particular activity identifier (it always impacts all active progress bars):

```
log_progress(
    lgr.info,
    None,
    'Clear progress bars',
    maint='clear',
)
```

This call will trigger a temporary discontinuation of any progress bar display. Progress bars can either be re-enabled all at once, by an analog message with `maint='refresh'`, or will re-show themselves automatically when the next update is received. A `no_progress()` context manager helper can be used to surround your context with those two calls to prevent progress bars from interfering.

GitHub Action

Specification scope and status

This specification describes a proposed interface to a DataLad GitHub Action. https://github.com/datalad/datalad-action provides an implementation which loosely followed this specification.

The purpose of the DataLad GitHub Action is to support CI testing with DataLad datasets by making it easy to install `datalad` and `get` data from the datasets.

Example Usage

Dataset installed at `${GITHUB_WORKSPACE}/studyforrest-data-phase2`, `get`'s all the data:

```
- uses: datalad/datalad-action@master
  with:
    datasets:
      - source: https://github.com/psychoinformatics-de/studyforrest-data-phase2
      - install_get_data: true
```

Specify advanced options:

```
- name: Download testing data
  uses: datalad/datalad-action@master
  with:
    datalad_version: ^0.15.5
    add_datalad_to_path: false
    datasets:
      - source: https://github.com/psychoinformatics-de/studyforrest-data-phase2
      - branch: develop
      - install_path: test_data
      - install_jobs: 2
      - install_get_data: false
      - recursive: true
      - recursion_limit: 2
      - get_jobs: 2
      - get_paths:
          - sub-01
          - sub-02
          - stimuli
```

Options

datalad_version

`datalad` version to install. Defaults to the latest release.

add_datalad_to_path

Add `datalad` to the `PATH` for manual invocation in subsequent steps.

Defaults to `true`.

source

URL for the dataset (mandatory).

branch

Git branch to install (optional).

install_path

Path to install the dataset relative to *GITHUB_WORKSPACE*.

Defaults to the repository name.

install_jobs

Jobs to use for `datalad install`.

Defaults to `auto`.

install_get_data

Get all the data in the dataset by passing `--get-data` to `datalad install`.

Defaults to `false`.

recursive

Boolean defining whether to clone subdatasets.

Defaults to `true`.

recursion_limit

Integer defining limits to recursion.

If not defined, there is no limit.

get_jobs

Jobs to use for `datalad get`.

Defaults to `auto`.

get_paths

A list of paths in the dataset to download with `datalad get`.

Defaults to everything.

Continuous integration and testing

Specification scope and status

This specification describes the current implementation.

DataLad is tested using a pytest-based testsuite that is run locally and via continuous integrations setups. Code development should ensure that old and new functionality is appropriately tested. The project aims for good unittest coverage (at least 80%).

Running tests

Starting at the top level with `datalad/tests`, every module in the package comes with a subdirectory `tests/`, containing the tests for that portion of the codebase. This structure is meant to simplify (re-)running the tests for a particular module. The test suite is run using

```
pip install -e .[tests]
python -m pytest -c tox.ini datalad
# or, with coverage reports
python -m pytest  -c tox.ini --cov=datalad datalad
```

Individual tests can be run using a path to the test file, followed by two colons and the test name:

```
python -m pytest datalad/core/local/tests/test_save.py::test_save_message_file
```

The set of to-be-run tests can be further sub-selected with environment variable based configurations that enable tests based on their *Test annotations*, or pytest-specific parameters. Invoking a test run using `DATALAD_TESTS_KNOWNFAILURES_PROBE=True pytest datalad`, for example, will run tests marked as known failures whether or not they still fail. See section *Configuration* for all available configurations. Invoking a test run using `DATALAD_TESTS_SSH=1 pytest -m xfail -c tox.ini datalad` will run only those tests marked as xfail.

Local setup

Local test execution usually requires a local installation with all development requirements. It is recommended to either use a virtualenv, or tox via a `tox.ini` file in the code base.

CI setup

At the moment, Travis-CI, Appveyor, and GitHub Workflows exercise the tests battery for every PR and on the default branch, covering different operating systems, Python versions, and file systems. Tests should be ran on the oldest, latest, and current stable Python release. The projects uses https://codecov.io for an overview of code coverage.

Writing tests

Additional functionality is tested by extending existing similar tests with new test cases, or adding new tests to the respective test script of the module. Generally, every file *example.py `with datalad code comes with a corresponding `tests/test_example.py*. Test helper functions assisting various general and DataLad specific assertions as well the construction of test directories and files can be found in `datalad/tests/utils_pytest.py`.

Test annotations

`datalad/tests/utils_pytest.py` also defines test decorators. Some of those are used to annotate tests for various aspects to allow for easy sub-selection via environment variables.

Speed: Please annotate tests that take a while to complete with following decorators

- `@slow` if test runs over 10 seconds
- `@turtle` if test runs over 120 seconds (those would not typically be ran on CIs)

Purpose: Please further annotate tests with a special purpose specifically. As those tests also usually tend to be slower, use in conjunction with `@slow` or `@turtle` when slow.

- `@integration` - tests verifying correct operation with external tools/services beyond git/git-annex
- `@usecase` - represents some (user) use-case, and not necessarily a "unit-test" of functionality

Dysfunction: If tests are not meant to be run on certain platforms or under certain conditions, `@known_failure` or `@skip` annotations can be used. Examples include:

- `@skip`, `@skip_if_on_windows`, `@skip_ssh`, `@skip_wo_symlink_capability`, `@skip_if_adjusted_branch`, `@skip_if_no_network`, `@skip_if_root`
- `@knownfailure`, `@known_failure_windows`, `known_failure_githubci_win` or `known_failure_githubci_osx`

Migrating tests from nose to pytest

DataLad's test suite has been migrated from nose to pytest in the 0.17.0 release. This might be relevant for DataLad extensions that still use nose.

For the time being, `datalad.tests.utils` keeps providing `nose`-based utils, and `datalad.__init__` keeps providing nose-based fixtures to not break extensions that still use nose for testing. A migration to `pytest` is recommended, though. To perform a typical migration of a DataLad extension to use pytest instead of nose, go through the following list:

- keep all the `assert_*` and `ok_` helpers, but import them from `datalad.tests.utils_pytest` instead
- for `@with_*` and other decorators populating positional arguments, convert corresponding posarg to kwarg by adding `=None`
- convert all generator-based parametric tests into direct invocations or, preferably, `@pytest.mark.parametrized` tests
- address `DeprecationWarnings` in the code. Only where desired to test deprecation, add `@pytest.mark.filterwarnings("ignore: BEGINNING OF WARNING")` decorator to the test.

For an example, see a "migrate to pytest" PR against `datalad-deprecated`: datalad/datalad-deprecated#51 .

User messaging: result records vs exceptions vs logging

Specification scope and status

This specification provides a partial overview of the implementation goal.

Motivation

This specification delineates the applicable contexts for using *result records*, *exceptions*, *progress reporting*, specific *log levels*, or other types of user messaging processes.

Specification

Result records

Result records are the only return value format for all DataLad interfaces.

Contrasting with classic Python interfaces that return specific non-annotated values, DataLad interfaces (i.e. subclasses of `datalad.interface.base.Interface`) implement message passing by yielding *result records* that are associated with individual operations. Result records are routinely inspected throughout the code base and their annotations are used to inform general program flow and error handling.

DataLad interface calls can include an `on_failure` parameterization to specify how to proceed with a particular operation if a returned result record is *classified as a failure result*. DataLad interface calls can also include a `result_renderer` parameterization to explicitly enable or disable the rendering of result records.

Developers should be aware that external callers will use DataLad interface call parameterizations that can selectively ignore or act on result records, and that the process should therefore yield meaningful result records. If, in turn, the process itself receives a set of result records from a sub-process, these should be inspected individually in order to identify result values that could require re-annotation or status re-classification.

For user messaging purposes, result records can also be enriched with additional human-readable information on the nature of the result, via the `message` key, and human-readable hints to the user, via the `hints` key. Both of these are rendered via the *UI Module*.

Exception handling

In general, **exceptions should be raised when there is no way to ignore or recover from the offending action**.

More specifically, raise an exception when:

1. A DataLad interface's parameter specifications are violated
2. An additional requirement (beyond parameters) for the meaningful continuation of a DataLad interface, function, or process is not met

It must be made clear to the user/caller what the exact cause of the exception is, given the context within which the user/caller triggered the action. This is achieved directly via a (re)raised exception, as opposed to logging messages or results records which could be ignored or unseen by the user.

Note: In the case of a complex set of dependent actions it could be expensive to confirm parameter violations. In such cases, initial sub-routines might already generate result records that have to be inspected by the caller, and it could be practically better to yield a result record (with `status=[error|impossible]`) to communicate the failure. It would then be up to the upstream caller to decide whether to specify `on_failure='ignore'` or whether to inspect individual result records and turn them into exceptions or not.

Logging

Logging provides developers with additional means to describe steps in a process, so as to **allow insight into the program flow during debugging** or analysis of e.g. usage patterns. Logging can be turned off externally, filtered, and redirected. Apart from the *log-level* and message, it is not inspectable and cannot be used to control the logic or flow of a program.

Importantly, logging should not be the primary user messaging method for command outcomes, Therefore:

1. No interface should rely solely on logging for user communication
2. Use logging for in-progress user communication via the mechanism for *progress reporting*
3. Use logging to inform debugging processes

UI Module

The `ui` module provides the means to communicate information to the user in a user-interface-specific manner, e.g. via a console, dialog, or an iPython interface. Internally, all DataLad results processed by the result renderer are passed through the UI module.

Therefore: unless the criteria for logging apply, and unless the message to be delivered to the user is specified via the `message` key of a result record, developers should let explicit user communication happen through the UI module as it provides the flexibility to adjust to the present UI. Specifically, `datalad.ui.message()` allows passing a simple message via the UI module.

Examples

The following links point to actual code implementations of the respective user messaging methods:

- Result yielding
- Exception handling
- Logging
- UI messaging

1.4.7 Glossary

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

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

CLI
: A Command Line Interface. Could be used interactively by executing commands in a shell, or as a programmable API for shell scripts.

DataLad extension
: A Python package, developed outside of the core DataLad codebase, which (when installed) typically either provides additional top level *datalad* commands and/or additional metadata extractors. Visit Handbook, Ch.2. DataLad's extensions for a representative list of extensions and instructions on how to install them.

dataset
: A regular Git repository with an (optional) *annex*.

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

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

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

1.5 Commands and API

1.5.1 Command line reference

Main command

datalad

Synopsis

```
datalad [-c (:name|name=value)] [-C PATH] [--cmd] [-l LEVEL] [--on-failure
    {ignore,continue,stop}] [--report-status
    {success,failure,ok,notneeded,impossible,error}] [--report-type
    {dataset,file}] [-f
    {generic,json,json_pp,tailored,disabled,'<template>'}] [--dbg]
    [--idbg] [--version] {create-sibling-github,create-sibling-gitla
    b,create-sibling-gogs,create-sibling-gin,create-sibling-gitea,cr
    eate-sibling-ria,create-sibling,siblings,update,subdatasets,drop
    ,remove,addurls,copy-file,download-url,foreach-dataset,install,r
    erun,run-procedure,create,save,status,clone,get,push,run,diff,co
    nfiguration,wtf,clean,add-archive-content,add-readme,export-arch
    ive,export-archive-ora,export-to-figshare,no-annex,check-dates,u
    nlock,uninstall,create-test-dataset,sshrun,shell-completion} ...
```

Description

Comprehensive data management solution

DataLad provides a unified data distribution system built on the Git and Git-annex. DataLad command line tools allow to manipulate (obtain, create, update, publish, etc.) datasets and provide a comprehensive toolbox for joint management of data and code. Compared to Git/annex it primarily extends their functionality to transparently and simultaneously work with multiple inter-related repositories.

Options

{create-sibling-github,create-sibling-gitlab,create-sibling-gogs,create-sibling-gin,create-sibling-gitea,create-sibling-ria,create-sibling,siblings,update,subdatasets,drop,remove,addurls,copy-file,download-url,foreach-dataset,install,rerun,run-procedure,create,save,status,clone,get,push,run,diff,configurati archive-content,add-readme,export-archive,export-archive-ora,export-to-figshare,no-annex,check-dates,unlock,uninstall,create-test-dataset,sshrun,shell-completion}

-c (:name|name=value)

specify configuration setting overrides. They override any configuration read from a file. A configuration can also be unset temporarily by prefixing its name with a colon (':'), e.g. ':user.name'. Overrides specified here may be overridden themselves by configuration settings declared as environment variables.

-C PATH

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

--cmd

syntactical helper that can be used to end the list of global command line options before the subcommand label. Options taking an arbitrary number of arguments may require to be followed by a single –cmd in order to enable identification of the subcommand.

-l LEVEL, --log-level LEVEL

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

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

when an operation fails: 'ignore' and continue with remaining operations, the error is logged but does not lead to a non-zero exit code of the command; 'continue' works like 'ignore', but an error causes a non-zero exit code; 'stop' halts on first failure and yields non-zero exit code. A failure is any result with status 'impossible' or 'error'. [Default: 'continue', but individual commands may define an alternative default]

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

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

--report-type {dataset,file}

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

-f {generic,json,json_pp,tailored,disabled,'<template>'}, --output-format {generic,json,json_pp,tailored,disabled,'<template>'}

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']

--dbg

enter Python debugger for an uncaught exception

--idbg

enter IPython debugger for an uncaught exception

--version

show the module and its version which provides the command

Authors

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

Core commands

A minimal set of commands that cover essential functionality. Core commands receive special scrutiny with regard API composition and (breaking) changes.

Local operation

datalad create

Synopsis

```
datalad create [-h] [-f] [-D DESCRIPTION] [-d DATASET] [--no-annex] [--fake-dates]
    [-c PROC] [--version] [PATH] ...
```

Description

Create a new dataset from scratch.

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

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

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

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

NOTE

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

Examples

Create a dataset 'mydataset' in the current directory:

```
% datalad create mydataset
```

Apply the text2git procedure upon creation of a dataset:

```
% datalad create -c text2git mydataset
```

Create a subdataset in the root of an existing dataset:

```
% datalad create -d . mysubdataset
```

Create a dataset in an existing, non-empty directory:

```
% datalad create --force
```

Create a plain Git repository:

```
% datalad create --no-annex mydataset
```

Options

PATH

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

INIT OPTIONS

options to pass to git init. Any argument specified after the destination path of the repository will be passed to git-init as-is. Note that not all options will lead to viable results. For example '–bare' will not yield a repository where DataLad can adjust files in its working tree.

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

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

-f, --force

enforce creation of a dataset in a non-empty directory.

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

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

-d DATASET, --dataset DATASET

specify the dataset to perform the create operation on. If a dataset is given along with PATH, a new subdataset will be created in it at the *path* provided to the create command. If a dataset is given but PATH is unspecified, a new dataset will be created at the location specified by this option. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--no-annex

if set, a plain Git repository will be created without any annex.

--fake-dates

Configure the repository to use fake dates. The date for a new commit will be set to one second later than the latest commit in the repository. This can be used to anonymize dates.

-c PROC, --cfg-proc PROC

Run cfg_PROC procedure(s) (can be specified multiple times) on the created dataset. Use run-procedure –discover to get a list of available procedures, such as cfg_text2git.

--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 save

Synopsis

```
datalad save [-h] [-m MESSAGE] [-d DATASET] [-t ID] [-r] [-R LEVELS] [-u] [-F
    MESSAGE_FILE] [--to-git] [-J NJOBS] [--amend] [--version] [PATH
    ...]
```

Description

Save the current state of a dataset

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

NOTE

Before Git v2.22, any Git repository without an initial commit located inside a Dataset is ignored, and content underneath it will be saved to the respective superdataset. DataLad datasets always have an initial commit, hence are not affected by this behavior.

Examples

Save any content underneath the current directory, without altering any potential subdataset:

```
% datalad save .
```

Save specific content in the dataset:

```
% datalad save myfile.txt
```

Attach a commit message to save:

```
% datalad save -m 'add file' myfile.txt
```

Save any content underneath the current directory, and recurse into any potential subdatasets:

```
% datalad save . -r
```

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

```
% datalad save -u .
```

Tag the most recent saved state of a dataset:

```
% datalad save --version-tag 'bestyet'
```

Save a specific change but integrate into last commit keeping the already recorded commit message:

```
% datalad save myfile.txt --amend
```

Options

PATH

path/name of the dataset component to save. If given, only changes made to those components are recorded in the new state. Constraints: value must be a string or value must be NONE

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

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

-m MESSAGE, --message MESSAGE

a description of the state or the changes made to a dataset. Constraints: value must be a string or value must be NONE

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

"specify the dataset to save. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-t ID, --version-tag ID

an additional marker for that state. Every dataset that is touched will receive the tag. Constraints: value must be a string or value must be NONE

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

-u, --updated

if given, only saves previously tracked paths.

-F *MESSAGE_FILE*, --message-file *MESSAGE_FILE*

take the commit message from this file. This flag is mutually exclusive with -m. Constraints: value must be a string or value must be NONE

--to-git

flag whether to add data directly to Git, instead of tracking data identity only. Use with caution, there is no guarantee that a file put directly into Git like this will not be annexed in a subsequent save operation. If not specified, it will be up to git-annex to decide how a file is tracked, based on a dataset's configuration to track particular paths, file types, or file sizes with either Git or git-annex. (see https://git-annex.branchable.com/tips/largefiles).

-J NJOBS, --jobs NJOBS

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

--amend

if set, changes are not recorded in a new, separate commit, but are integrated with the changeset of the previous commit, and both together are recorded by replacing that previous commit. This is mutually exclusive with recursive operation.

--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 run

Synopsis

```
datalad run [-h] [-d DATASET] [-i PATH] [-o PATH] [--expand {inputs|outputs|both}]
    [--assume-ready {inputs|outputs|both}] [--explicit] [-m MESSAGE]
    [--sidecar {yes|no}] [--dry-run {basic|command}] [-J NJOBS]
    [--version] ...
```

Description

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

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

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

If the given command errors, a COMMANDERROR exception with the same exit code will be raised, and no modifications will be saved. A command execution will not be attempted, by default, when an error occurred during input or output preparation. This default `stop` behavior can be overridden via –on-failure ....

In the presence of subdatasets, the full dataset hierarchy will be checked for unsaved changes prior command execution, and changes in any dataset will be saved after execution. Any modification of subdatasets is also saved in their respective superdatasets to capture a comprehensive record of the entire dataset hierarchy state. The associated provenance record is duplicated in each modified (sub)dataset, although only being fully interpretable and re-executable in the actual top-level superdataset. For this reason the provenance record contains the dataset ID of that superdataset.

Command format

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

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

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

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

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

```
% datalad configuration --scope branch set datalad.run.substitutions.name=joe
% datalad save -m "Configure name placeholder" .datalad/config
```

Access the new placeholder in a command:

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

Examples

Run an executable script and record the impact on a dataset:

```
% datalad run -m 'run my script' 'code/script.sh'
```

Run a command and specify a directory as a dependency for the run. The contents of the dependency will be retrieved prior to running the script:

```
% datalad run -m 'run my script' -i 'data/*' 'code/script.sh'
```

Run an executable script and specify output files of the script to be unlocked prior to running the script:

```
% datalad run -m 'run my script' -i 'data/*' \
  -o 'output_dir/*' 'code/script.sh'
```

Specify multiple inputs and outputs:

```
% datalad run -m 'run my script' -i 'data/*' \
  -i 'datafile.txt' -o 'output_dir/*' -o \
  'outfile.txt' 'code/script.sh'
```

Use ** to match any file at any directory depth recursively. Single * does not check files within matched directories.:

```
% datalad run -m 'run my script' -i 'data/**/*.dat' \
  -o 'output_dir/**' 'code/script.sh'
```

Options

COMMAND

command for execution. A leading '–' can be used to disambiguate this command from the preceding options to DataLad.

-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 record the command results in. An attempt is made to identify the dataset based on the current working directory. If a dataset is given, the command will be executed in the root directory of this dataset. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-i PATH, --input PATH

A dependency for the run. Before running the command, the content for this relative path will be retrieved. A value of "." means "run datalad get .". The value can also be a glob. This option can be given more than once.

-o PATH, --output PATH

Prepare this relative path to be an output file of the command. A value of "." means "run datalad unlock ." (and will fail if some content isn't present). For any other value, if the content of this file is present, unlock the file. Otherwise, remove it. The value can also be a glob. This option can be given more than once.

--expand {inputs|outputs|both}

Expand globs when storing inputs and/or outputs in the commit message. Constraints: value must be one of ('inputs', 'outputs', 'both')

--assume-ready {inputs|outputs|both}

Assume that inputs do not need to be retrieved and/or outputs do not need to unlocked or removed before running the command. This option allows you to avoid the expense of these preparation steps if you know that they are unnecessary. Constraints: value must be one of ('inputs', 'outputs', 'both')

--explicit

Consider the specification of inputs and outputs to be explicit. Don't warn if the repository is dirty, and only save modifications to the listed outputs.

-m MESSAGE, --message MESSAGE

a description of the state or the changes made to a dataset. Constraints: value must be a string or value must be NONE

--sidecar {yes|no}

By default, the configuration variable 'datalad.run.record-sidecar' determines whether a record with information on a command's execution is placed into a separate record file instead of the commit message (default: off). This option can be used to override the configured behavior on a case-by-case basis. Sidecar files are placed into the dataset's '.datalad/runinfo' directory (customizable via the 'datalad.run.record-directory' configuration variable). Constraints: value must be NONE or value must be convertible to type bool

--dry-run {basic|command}

Do not run the command; just display details about the command execution. A value of "basic" reports a few important details about the execution, including the expanded command and expanded inputs and outputs. "command" displays the expanded command only. Note that input and output globs underneath an uninstalled dataset will be left unexpanded because no subdatasets will be installed for a dry run. Constraints: value must be one of ('basic', 'command')

-J NJOBS, --jobs NJOBS

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

--version

show the module and its version which provides the command

Authors

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

datalad status

Synopsis

```
datalad status [-h] [-d DATASET] [--annex [{basic|availability|all}]] [--untracked
    {no|normal|all}] [-r] [-R LEVELS] [-e {no|commit|full}] [-t
    {raw|eval}] [--version] [PATH ...]
```

Description

Report on the state of dataset content.

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

Path conventions

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

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

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

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

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

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

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

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

Content types

The following content types are distinguished:

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

Content states

The following content states are distinguished:

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

Examples

Report on the state of a dataset:

```
% datalad status
```

Report on the state of a dataset and all subdatasets:

```
% datalad status -r
```

Address a subdataset record in a superdataset without causing a status query for the state _within_ the subdataset itself:

```
% datalad status -d . mysubdataset
```

Get a status query for the state within the subdataset without causing a status query for the superdataset (using trailing path separator in the query path)::

```
% datalad status -d . mysubdataset/
```

Report on the state of a subdataset in a superdataset and on the state within the subdataset:

```
% datalad status -d . mysubdataset mysubdataset/
```

Report the file size of annexed content in a dataset:

```
% datalad status --annex
```

Options

PATH

path to be evaluated. Constraints: value must be a string or value must be NONE

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

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

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

specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--annex [{basic|availability|all}]

Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). The 'basic' mode will be assumed when this option is given, but no mode is specified. Constraints: value must be one of ('basic', 'availability', 'all')

--untracked {no|normal|all}

If and how untracked content is reported when comparing a revision to the state of the working tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. Constraints: value must be one of ('no', 'normal', 'all') [Default: 'normal']

-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

-e {no|commit|full}, --eval-subdataset-state {no|commit|full}

Evaluation of subdataset state (clean vs. modified) can be expensive for deep dataset hierarchies as subdataset have to be tested recursively for uncommitted modifications. Setting this option to 'no' or 'commit' can substantially boost performance by limiting what is being tested. With 'no' no state is evaluated and subdataset result records typically do not contain a 'state' property. With 'commit' only a discrepancy of the HEAD commit shasum of a subdataset and the shasum recorded in the superdataset's record is evaluated, and the 'state' result property only reflects this aspect. With 'full' any other modification is considered too (see the 'untracked' option for further tailoring modification testing). Constraints: value must be one of ('no', 'commit', 'full') [Default: 'full']

-t {raw|eval}, --report-filetype {raw|eval}

THIS OPTION IS IGNORED. It will be removed in a future release. Dataset component types are always reported as-is (previous 'raw' mode), unless annex- reporting is enabled with the –annex option, in which case symlinks that represent annexed files will be reported as type='file'. Constraints: value must be one of ('raw', 'eval')

--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 diff

Synopsis

```
datalad diff [-h] [-f REVISION] [-t REVISION] [-d DATASET] [--annex
    [{basic|availability|all}]] [--untracked {no|normal|all}] [-r]
    [-R LEVELS] [--version] [PATH ...]
```

Description

Report differences between two states of a dataset (hierarchy)

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

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

Reports are very similar to those of the STATUS command, with the distinguished content types and states being identical.

Examples

Show unsaved changes in a dataset:

```
% datalad diff
```

Compare a previous dataset state identified by shasum against current worktree:

```
% datalad diff --from <SHASUM>
```

Compare two branches against each other:

```
% datalad diff --from branch1 --to branch2
```

Show unsaved changes in the dataset and potential subdatasets:

```
% datalad diff -r
```

Show unsaved changes made to a particular file:

```
% datalad diff <path/to/file>
```

Options

PATH

path to constrain the report to. Constraints: value must be a string or value must be NONE

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

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

-f REVISION, --from REVISION

original state to compare to, as given by any identifier that Git understands. Constraints: value must be a string [Default: 'HEAD']

-t REVISION, --to REVISION

state to compare against the original state, as given by any identifier that Git understands. If none is specified, the state of the working tree will be compared. Constraints: value must be a string or value must be NONE

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

specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--annex [{basic|availability|all}]

Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). The 'basic' mode will be assumed when this option is given, but no mode is specified. Constraints: value must be one of ('basic', 'availability', 'all')

--untracked {no|normal|all}

If and how untracked content is reported when comparing a revision to the state of the working tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. Constraints: value must be one of ('no', 'normal', 'all') [Default: 'normal']

-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>.

Distributed operation

datalad clone

Synopsis

```
datalad clone [-h] [-d DATASET] [-D DESCRIPTION] [--reckless
    [auto|ephemeral|shared-...]] [--version] SOURCE [PATH] ...
```

Description

Obtain a dataset (copy) from a URL or local directory

The purpose of this command is to obtain a new clone (copy) of a dataset and place it into a not-yet-existing or empty directory. As such CLONE provides a strict subset of the functionality offered by *install*. Only a single dataset can be obtained, and immediate recursive installation of subdatasets is not supported. However, once a (super)dataset is installed via CLONE, any content, including subdatasets can be obtained by a subsequent *get* command.

Primary differences over a direct *git clone* call are 1) the automatic initialization of a dataset annex (pure Git repositories are equally supported); 2) automatic registration of the newly obtained dataset as a subdataset (submodule), if a parent dataset is specified; 3) support for additional resource identifiers (DataLad resource identifiers as used on datasets.datalad.org, and RIA store URLs as used for store.datalad.org - optionally in specific versions as identified by

a branch or a tag; see examples); and 4) automatic configurable generation of alternative access URL for common cases (such as appending '.git' to the URL in case the accessing the base URL failed).

In case the clone is registered as a subdataset, the original URL passed to CLONE is recorded in *.gitmodules* of the parent dataset in addition to the resolved URL used internally for git-clone. This allows to preserve datalad specific URLs like ria+ssh://... for subsequent calls to GET if the subdataset was locally removed later on.

URL mapping configuration

'clone' supports the transformation of URLs via (multi-part) substitution specifications. A substitution specification is defined as a configuration setting 'datalad.clone.url-substition.<seriesID>' with a string containing a match and substitution expression, each following Python's regular expression syntax. Both expressions are concatenated to a single string with an arbitrary delimiter character. The delimiter is defined by prefixing the string with the delimiter. Prefix and delimiter are stripped from the expressions (Example: ",^http://(.*)$,https://1"). This setting can be defined multiple times, using the same '<seriesID>'. Substitutions in a series will be applied incrementally, in order of their definition. The first substitution in such a series must match, otherwise no further substitutions in a series will be considered. However, following the first match all further substitutions in a series are processed, regardless whether intermediate expressions match or not. Substitution series themselves have no particular order, each matching series will result in a candidate clone URL. Consequently, the initial match specification in a series should be as precise as possible to prevent inflation of candidate URLs.

SEEALSO

handbook:3-001 (http://handbook.datalad.org/symbols)
More information on Remote Indexed Archive (RIA) stores

Examples

Install a dataset from GitHub into the current directory:

```
% datalad clone https://github.com/datalad-datasets/longnow-podcasts.git
```

Install a dataset into a specific directory:

```
% datalad clone https://github.com/datalad-datasets/longnow-podcasts.git \
  myfavpodcasts
```

Install a dataset as a subdataset into the current dataset:

```
% datalad clone -d . https://github.com/datalad-datasets/longnow-podcasts.git
```

Install the main superdataset from datasets.datalad.org:

```
% datalad clone ///
```

Install a dataset identified by a literal alias from store.datalad.org:

```
% datalad clone ria+http://store.datalad.org#~hcp-openaccess
```

Install a dataset in a specific version as identified by a branch or tag name from store.datalad.org:

```
% datalad clone ria+http://store.datalad.org#76b6ca66-36b1-11ea-a2e6-
↪f0d5bf7b5561@myidentifier
```

Install a dataset with group-write access permissions:

```
% datalad clone http://example.com/dataset --reckless shared-group
```

Options

SOURCE

URL, DataLad resource identifier, local path or instance of dataset to be cloned. Constraints: value must be a string

PATH

path to clone into. If no PATH is provided a destination path will be derived from a source URL similar to git clone.

GIT CLONE OPTIONS

Options to pass to git clone. Any argument specified after SOURCE and the optional PATH will be passed to git-clone. Note that not all options will lead to viable results. For example '–single-branch' will not result in a functional annex repository because both a regular branch and the git-annex branch are required. Note that a version in a RIA URL takes precedence over '–branch'.

-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*

(parent) dataset to clone into. If given, the newly cloned dataset is registered as a subdataset of the parent. Also, if given, relative paths are interpreted as being relative to the parent dataset, and not relative to the working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

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

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

--reckless [auto|ephemeral|shared-...]

Obtain a dataset or subdatset and set it up in a potentially unsafe way for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git- annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. Constraints: value must be one of (True, False, 'auto', 'ephemeral') or value must start with 'shared-'

--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 push

Synopsis

```
datalad push [-h] [-d DATASET] [--to SIBLING] [--since SINCE] [--data
    {anything|nothing|auto|auto-if-wanted}] [-f
    {all|gitpush|checkdatapresent}] [-r] [-R LEVELS] [-J NJOBS]
    [--version] [PATH ...]
```

Description

Push a dataset to a known sibling.

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

By default, all files tracked in the last saved state (of the current branch) will be copied to the target location. Optionally, it is possible to limit a push to changes relative to a particular point in the version history of a dataset (e.g. a release tag) using the –since option in conjunction with the specification of a reference dataset. In recursive mode subdatasets will also be evaluated, and only those subdatasets are pushed where a change was recorded that is reflected in the current state of the top-level reference dataset.

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

Options

PATH

path to constrain a push to. If given, only data or changes for those paths are considered for a push. Constraints: value must be a string or value must be NONE

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

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

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

specify the dataset to push. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--to SIBLING

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

--since *SINCE*

specifies commit-ish (tag, shasum, etc.) from which to look for changes to decide whether pushing is necessary. If '^' is given, the last state of the current branch at the sibling is taken as a starting point. Constraints: value must be a string or value must be NONE

--data {anything|nothing|auto|auto-if-wanted}

what to do with (annex'ed) data. 'anything' would cause transfer of all annexed content, 'nothing' would avoid call to *git annex copy* altogether. 'auto' would use 'git annex copy' with '–auto' thus transferring only data which would satisfy "wanted" or "numcopies" settings for the remote (thus "nothing" otherwise). 'auto-if-wanted' would enable '–auto' mode only if there is a "wanted" setting for the remote, and transfer 'anything' otherwise. Constraints: value must be one of ('anything', 'nothing', 'auto', 'auto-if-wanted') [Default: 'auto-if-wanted']

-f {all|gitpush|checkdatapresent}, --force {all|gitpush|checkdatapresent}

force particular operations, possibly overruling safety protections or optimizations: use –force with git-push ('gitpush'); do not use –fast with git-annex copy ('checkdatapresent'); combine all force modes ('all'). Constraints: value must be one of ('all', 'gitpush', 'checkdatapresent')

-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

-J NJOBS, --jobs NJOBS

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

--version

show the module and its version which provides the command

Authors

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

Extended set of functionality

Dataset operations

datalad add-readme

Synopsis

```
datalad add-readme [-h] [-d DATASET] [--existing {skip|append|replace}] [--version]
    [PATH]
```

Description

Add basic information about DataLad datasets to a README file

The README file is added to the dataset and the addition is saved in the dataset. Note: Make sure that no unsaved modifications to your dataset's .gitattributes file exist.

Options

PATH

Path of the README file within the dataset. Constraints: value must be a string [Default: 'README.md']

-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 add information to. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--existing {skip|append|replace}

How to react if a file with the target name already exists: 'skip': do nothing; 'append': append information to the existing file; 'replace': replace the existing file with new content. Constraints: value must be one of ('skip', 'append', 'replace') [Default: 'skip']

--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 addurls

Synopsis

```
datalad addurls [-h] [-d DATASET] [-t TYPE] [-x REGEXP] [-m FORMAT] [--key FORMAT]
    [--message MESSAGE] [-n] [--fast] [--ifexists {overwrite|skip}]
    [--missing-value VALUE] [--nosave] [--version-urls] [-c PROC]
    [-J NJOBS] [--drop-after] [--on-collision
    {error|error-if-different|take-first|take-last}] [--version]
    URL-FILE URL-FORMAT FILENAME-FORMAT
```

Description

Create and update a dataset from a list of URLs.

Format specification

Several arguments take format strings. These are similar to normal Python format strings where the names from *URL-FILE* (column names for a comma- or tab-separated file or properties for JSON) are available as placeholders. If *URL-FILE* is a CSV or TSV file, a positional index can also be used (i.e., "{0}" for the first column). Note that a placeholder cannot contain a ':' or '!'.

In addition, the *FILENAME-FORMAT* arguments has a few special placeholders.

- _repindex

  The constructed file names must be unique across all fields rows. To avoid collisions, the special placeholder "_repindex" can be added to the formatter. Its value will start at 0 and increment every time a file name repeats.

- _url_hostname, _urlN, _url_basename*

  Various parts of the formatted URL are available. Take "http://datalad.org/asciicast/seamless_nested_repos.sh" as an example.

  "datalad.org" is stored as "_url_hostname". Components of the URL's path can be referenced as "_urlN". "_url0" and "_url1" would map to "asciicast" and "seamless_nested_repos.sh", respectively. The final part of the path is also available as "_url_basename".

  This name is broken down further. "_url_basename_root" and "_url_basename_ext" provide access to the root name and extension. These values are similar to the result of os.path.splitext, but, in the case of multiple periods, the extension is identified using the same length heuristic that git-annex uses. As a result, the extension of "file.tar.gz" would be ".tar.gz", not ".gz". In addition, the fields "_url_basename_root_py" and "_url_basename_ext_py" provide access to the result of os.path.splitext.

- _url_filename*

  These are similar to _url_basename* fields, but they are obtained with a server request. This is useful if the file name is set in the Content-Disposition header.

Examples

Consider a file "avatars.csv" that contains:

```
who,ext,link
neurodebian,png,https://avatars3.githubusercontent.com/u/260793
datalad,png,https://avatars1.githubusercontent.com/u/8927200
```

To download each link into a file name composed of the 'who' and 'ext' fields, we could run:

```
$ datalad addurls -d avatar_ds avatars.csv '{link}' '{who}.{ext}'
```

The *-d avatar_ds* is used to create a new dataset in "$PWD/avatar_ds".

If we were already in a dataset and wanted to create a new subdataset in an "avatars" subdirectory, we could use "//" in the *FILENAME-FORMAT* argument:

```
$ datalad addurls avatars.csv '{link}' 'avatars//{who}.{ext}'
```

If the information is represented as JSON lines instead of comma separated values or a JSON array, you can use a utility like jq to transform the JSON lines into an array that addurls accepts:

```
$ ... | jq --slurp . | datalad addurls - '{link}' '{who}.{ext}'
```

NOTE

For users familiar with 'git annex addurl': A large part of this plugin's functionality can be viewed as transforming data from *URL-FILE* into a "url filename" format that fed to 'git annex addurl –batch –with-files'.

Options

URL-FILE

A file that contains URLs or information that can be used to construct URLs. Depending on the value of –input-type, this should be a comma- or tab-separated file (with a header as the first row) or a JSON file (structured as a list of objects with string values). If '-', read from standard input, taking the content as JSON when –input-type is at its default value of 'ext'.

URL-FORMAT

A format string that specifies the URL for each entry. See the 'Format Specification' section above.

FILENAME-FORMAT

Like *URL-FORMAT*, but this format string specifies the file to which the URL's content will be downloaded. The name should be a relative path and will be taken as relative to the top-level dataset, regardless of whether it is specified via –dataset or inferred. The file name may contain directories. The separator "//" can be used to indicate that the left-side directory should be created as a new subdataset. See the 'Format Specification' section above.

-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*

Add the URLs to this dataset (or possibly subdatasets of this dataset). An empty or non-existent directory is passed to create a new dataset. New subdatasets can be specified with *FILENAME-FORMAT*. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-t TYPE, --input-type TYPE

Whether *URL-FILE* should be considered a CSV file, TSV file, or JSON file. The default value, "ext", means to consider *URL-FILE* as a JSON file if it ends with ".json" or a TSV file if it ends with ".tsv". Otherwise, treat it as a CSV file. Constraints: value must be one of ('ext', 'csv', 'tsv', 'json') [Default: 'ext']

-x REGEXP, --exclude-autometa REGEXP

By default, metadata field=value pairs are constructed with each column in *URL- FILE*, excluding any single column that is specified via *URL-FORMAT*. This argument can be used to exclude columns that match a regular expression. If set to '*' or an empty string, automatic metadata extraction is disabled completely. This argument does not affect metadata set explicitly with –meta.

-m FORMAT, --meta FORMAT

A format string that specifies metadata. It should be structured as "<field>=<value>". As an example, "location={3}" would mean that the value for the "location" metadata field should be set the value of the fourth column. This option can be given multiple times.

--key FORMAT

A format string that specifies an annex key for the file content. In this case, the file is not downloaded; instead the key is used to create the file without content. The value should be structured as "[et:]<input backend>[-s<bytes>]–<hash>". The optional "et:" prefix, which requires git- annex 8.20201116 or later, signals to toggle extension state of the input backend (i.e., MD5 vs MD5E). As an example, "et:MD5-s{size}–{md5sum}" would use the 'md5sum' and 'size' columns to construct the key, migrating the key from MD5 to MD5E, with an extension based on the file name. Note: If the *input* backend itself is an annex extension backend (i.e., a backend with a trailing "E"), the key's extension will not be updated to match the extension of the corresponding file name. Thus, unless the input keys and file names are generated from git- annex, it is recommended to avoid using extension backends as input. If an extension is desired, use the plain variant as input and prepend "et:" so that git-annex will migrate from the plain backend to the extension variant.

--message MESSAGE

Use this message when committing the URL additions. Constraints: value must be NONE or value must be a string

-n, --dry-run

Report which URLs would be downloaded to which files and then exit.

--fast

If True, add the URLs, but don't download their content. WARNING: ONLY USE THIS OPTION IF YOU UNDERSTAND THE CONSEQUENCES. If the content of the URLs is not downloaded, then datalad will refuse to retrieve the contents with *datalad get <file>* by default because the content of the URLs is not verified. Add *annex.security.allow-unverified-downloads = ACKTHPPT* to your git config to bypass the safety check. Underneath, this passes the *–fast* flag to *git annex addurl*.

--ifexists {overwrite|skip}

What to do if a constructed file name already exists. The default behavior is to proceed with the *git annex addurl*, which will fail if the file size has changed. If set to 'overwrite', remove the old file before adding the new one. If set to 'skip', do not add the new file. Constraints: value must be one of ('overwrite', 'skip')

--missing-value VALUE

When an empty string is encountered, use this value instead. Constraints: value must be NONE or value must be a string

--nosave

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

--version-urls

Try to add a version ID to the URL. This currently only has an effect on HTTP URLs for AWS S3 buckets. s3:// URL versioning is not yet supported, but any URL that already contains a "versionId=" parameter will be used as is.

-c PROC, --cfg-proc PROC

Pass this –cfg_proc value when calling CREATE to make datasets.

-J NJOBS, --jobs NJOBS

how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. Constraints: value must be convertible to type 'int' or value must be NONE or value must be one of ('auto',)

--drop-after

drop files after adding to annex.

--on-collision {error|error-if-different|take-first|take-last}

What to do when more than one row produces the same file name. By default an error is triggered. "error-if-different" suppresses that error if rows for a given file name collision have the same URL and metadata. "take-first" or "take-last" indicate to instead take the first row or last row from each set of colliding rows. Constraints: value must be one of ('error', 'error-if- different', 'take-first', 'take-last') [Default: 'error']

--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 copy-file

Synopsis

```
datalad copy-file [-h] [-d DATASET] [--recursive] [--target-dir DIRECTORY] [--specs-from
    SOURCE] [-m MESSAGE] [--version] [PATH ...]
```

Description

Copy files and their availability metadata from one dataset to another.

The difference to a system copy command is that here additional content availability information, such as registered URLs, is also copied to the target dataset. Moreover, potentially required git-annex special remote configurations are detected in a source dataset and are applied to a target dataset in an analogous fashion. It is possible to copy a file for which no content is available locally, by just copying the required metadata on content identity and availability.

NOTE
At the moment, only URLs for the special remotes 'web' (git-annex built-in) and 'datalad' are recognized and transferred.

The interface is modeled after the POSIX 'cp' command, but with one additional way to specify what to copy where: –specs-from allows the caller to flexibly input source-destination path pairs.

This command can copy files out of and into a hierarchy of nested datasets. Unlike with other DataLad command, the –recursive switch does not enable recursion into subdatasets, but is analogous to the POSIX 'cp' command switch and enables subdirectory recursion, regardless of dataset boundaries. It is not necessary to enable recursion in order to save changes made to nested target subdatasets.

Examples

Copy a file into a dataset 'myds' using a path and a target directory specification, and save its addition to 'myds':

```
% datalad copy-file path/to/myfile -d path/to/myds
```

Copy a file to a dataset 'myds' and save it under a new name by providing two paths:

```
% datalad copy-file path/to/myfile path/to/myds/new -d path/to/myds
```

Copy a file into a dataset without saving it:

```
% datalad copy-file path/to/myfile -t path/to/myds
```

Copy a directory and its subdirectories into a dataset 'myds' and save the addition in 'myds':

```
% datalad copy-file path/to/dir -r -d path/to/myds
```

Copy files using a path and optionally target specification from a file:

```
% datalad copy-file -d path/to/myds --specs-from specfile
```

Read a specification from stdin and pipe the output of a find command into the copy-file command:

```
% find <expr> | datalad copy-file -d path/to/myds --specs-from -
```

Options

PATH

paths to copy (and possibly a target path to copy to). Constraints: value must be a string or value must be NONE

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

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

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

root dataset to save after copy operations are completed. All destination paths must be within this dataset, or its subdatasets. If no dataset is given, dataset modifications will be left unsaved. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--recursive, -r

copy directories recursively.

--target-dir DIRECTORY, -t DIRECTORY

copy all source files into this DIRECTORY. This value is overridden by any explicit destination path provided via –specs-from. When not given, this defaults to the path of the dataset specified via –dataset. Constraints: value must be a string or value must be NONE

--specs-from SOURCE

read list of source (and destination) path names from a given file, or stdin (with '-'). Each line defines either a source path, or a source/destination path pair (separated by a null byte character).

-m MESSAGE, --message MESSAGE

a description of the state or the changes made to a dataset. Constraints: value must be a string 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 drop

Synopsis

```
datalad drop [-h] [--what {filecontent|allkeys|datasets|all}] [--reckless
    {modification|availability|undead|kill}] [-d DATASET] [-r] [-R
    LEVELS] [-J NJOBS] [--nocheck] [--if-dirty IF_DIRTY] [--version]
    [PATH ...]
```

Description

Drop content of individual files or entire (sub)datasets

This command is the antagonist of 'get'. It can undo the retrieval of file content, and the installation of subdatasets.

Dropping is a safe-by-default operation. Before dropping any information, the command confirms the continued availability of file-content (see e.g., configuration 'annex.numcopies'), and the state of all dataset branches from at least one known dataset sibling. Moreover, prior removal of an entire dataset annex, that it is confirmed that it is no longer marked as existing in the network of dataset siblings.

Importantly, all checks regarding version history availability and local annex availability are performed using the current state of remote siblings as known to the local dataset. This is done for performance reasons and for resilience in case of absent network connectivity. To ensure decision making based on up-to-date information, it is advised to execute a dataset update before dropping dataset components.

Examples

Drop single file content:

```
% datalad drop <path/to/file>
```

Drop all file content in the current dataset:

```
% datalad drop
```

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

```
% datalad drop -d <path/to/dataset> -r
```

Disable check to ensure the configured minimum number of remote sources for dropped data:

```
% datalad drop <path/to/content> --reckless availability
```

Drop (uninstall) an entire dataset (will fail with subdatasets present):

```
% datalad drop --what all
```

Kill a dataset recklessly with any existing subdatasets too(this will be fast, but will disable any and all safety checks):

```
% datalad drop --what all, --reckless kill --recursive
```

Options

PATH

path of a dataset or dataset component to be dropped. Constraints: value must be a string or value must be NONE

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

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

--what {filecontent|allkeys|datasets|all}

select what type of items shall be dropped. With 'filecontent', only the file content (git-annex keys) of files in a dataset's worktree will be dropped. With 'allkeys', content of any version of any file in any branch (including, but not limited to the worktree) will be dropped. This effectively empties the annex of a local dataset. With 'datasets', only complete datasets will be dropped (implies 'allkeys' mode for each such dataset), but no filecontent will be dropped for any files in datasets that are not dropped entirely. With 'all', content for any matching file or dataset will be dropped entirely. Constraints: value must be one of ('filecontent', 'allkeys', 'datasets', 'all') [Default: 'filecontent']

--reckless {modification|availability|undead|kill}

disable individual or all data safety measures that would normally prevent potentially irreversible data-loss. With 'modification', unsaved modifications in a dataset will not be detected. This improves performance at the cost of permitting potential loss of unsaved or untracked dataset components. With 'availability', detection of dataset/branch-states that are only available in the local dataset, and detection of an insufficient number of file-content copies will be disabled. Especially the latter is a potentially expensive check which might involve numerous network transactions. With 'undead', detection of whether a to-be-removed local annex is still known to exist in the network of dataset-clones is disabled. This could cause zombie-records of invalid file availability. With 'kill', all safety-checks are disabled. Constraints: value must be one of ('modification', 'availability', 'undead', 'kill')

-d DATASET, --dataset DATASET

specify the dataset to perform drop from. If no dataset is given, the current working directory is used as operation context. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

-J NJOBS, --jobs NJOBS

how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. Constraints: value must be convertible to type 'int' or value must be NONE or value must be one of ('auto',)

--nocheck

DEPRECATED: use '–reckless availability'.

--if-dirty *IF_DIRTY*

DEPRECATED and IGNORED: use –reckless instead.

--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 get

Synopsis

```
datalad get [-h] [-s LABEL] [-d PATH] [-r] [-R LEVELS] [-n] [-D DESCRIPTION]
    [--reckless [auto|ephemeral|shared-...]] [-J NJOBS] [--version]
    [PATH ...]
```

Description

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

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

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

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

Getting subdatasets

Just as DataLad supports getting file content from more than one location, the same is supported for subdatasets, including a ranking of individual sources for prioritization.

The following location candidates are considered. For each candidate a cost is given in parenthesis, higher values indicate higher cost, and thus lower priority:

- A datalad URL recorded in *.gitmodules* (cost 590). This allows for datalad URLs that require additional handling/resolution by datalad, like ria-schemes (ria+http, ria+ssh, etc.)
- A URL or absolute path recorded for git in *.gitmodules* (cost 600).
- URL of any configured superdataset remote that is known to have the desired submodule commit, with the submodule path appended to it. There can be more than one candidate (cost 650).
- In case *.gitmodules* contains a relative path instead of a URL, the URL of any configured superdataset remote that is known to have the desired submodule commit, with this relative path appended to it. There can be more than one candidate (cost 650).
- In case *.gitmodules* contains a relative path as a URL, the absolute path of the superdataset, appended with this relative path (cost 900).

Additional candidate URLs can be generated based on templates specified as configuration variables with the pattern

datalad.get.subdataset-source-candidate-<name>

where NAME is an arbitrary identifier. If *name* starts with three digits (e.g. '400myserver') these will be interpreted as a cost, and the respective candidate will be sorted into the generated candidate list according to this cost. If no cost is given, a default of 700 is used.

A template string assigned to such a variable can utilize the Python format mini language and may reference a number of properties that are inferred from the parent dataset's knowledge about the target subdataset. Properties include any submodule property specified in the respective *.gitmodules* record. For convenience, an existing *datalad-id* record is made available under the shortened name ID.

Additionally, the URL of any configured remote that contains the respective submodule commit is available as *remoteurl-<name>* property, where NAME is the configured remote name.

Hence, such a template could be *http://example.org/datasets/{id}* or *http://example.org/datasets/{path}*, where *{id}* and *{path}* would be replaced by the *datalad-id* or PATH entry in the *.gitmodules* record.

If this config is committed in *.datalad/config*, a clone of a dataset can look up any subdataset's URL according to such scheme(s) irrespective of what URL is recorded in *.gitmodules*.

Lastly, all candidates are sorted according to their cost (lower values first), and duplicate URLs are stripped, while preserving the first item in the candidate list.

NOTE

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

Examples

Get a single file:

```
% datalad get <path/to/file>
```

Get contents of a directory:

```
% datalad get <path/to/dir/>
```

Get all contents of the current dataset and its subdatasets:

```
% datalad get . -r
```

Get (clone) a registered subdataset, but don't retrieve data:

```
% datalad get -n <path/to/subds>
```

Options

PATH

path/name of the requested dataset component. The component must already be known to a dataset. To add new components to a dataset use the ADD command. Constraints: value must be a string or value must be NONE

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

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

-s LABEL, --source LABEL

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

-d PATH, --dataset PATH

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

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

-n, --no-data

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

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

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

--reckless [auto|ephemeral|shared-...]

Obtain a dataset or subdatset and set it up in a potentially unsafe way for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git- annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. Constraints: value must be one of (True, False, 'auto', 'ephemeral') or value must start with 'shared-'

-J NJOBS, --jobs NJOBS

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

--version

show the module and its version which provides the command

Authors

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

datalad install

Synopsis

```
datalad install [-h] [-s URL-OR-PATH] [-d DATASET] [-g] [-D DESCRIPTION] [-r] [-R
    LEVELS] [--reckless [auto|ephemeral|shared-...]] [-J NJOBS]
    [--branch BRANCH] [--version] [URL-OR-PATH ...]
```

Description

Install one or many datasets from remote URL(s) or local PATH source(s).

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

If no explicit -s|–source option is specified, then all positional URL-OR-PATH arguments are considered to be "sources" if they are URLs or target locations if they are paths. If a target location path corresponds to a submodule, the source location for it is figured out from its record in the *.gitmodules*. If -s|–source is specified, then a single optional positional PATH would be taken as the destination path for that dataset.

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

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

NOTE

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

Examples

Install a dataset from GitHub into the current directory:

```
% datalad install https://github.com/datalad-datasets/longnow-podcasts.git
```

Install a dataset as a subdataset into the current dataset:

```
% datalad install -d . \
  --source='https://github.com/datalad-datasets/longnow-podcasts.git'
```

Install a dataset into 'podcasts' (not 'longnow-podcasts') directory, and get all content right away:

```
% datalad install --get-data \
  -s https://github.com/datalad-datasets/longnow-podcasts.git podcasts
```

Install a dataset with all its subdatasets:

```
% datalad install -r \
  https://github.com/datalad-datasets/longnow-podcasts.git
```

Options

URL-OR-PATH

path/name of the installation target. If no PATH is provided a destination path will be derived from a source URL similar to git clone.

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

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

-s URL-OR-PATH, --source URL-OR-PATH

URL or local path of the installation source. Constraints: value must be a string or value must be NONE

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

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

-g, --get-data

if given, obtain all data content too.

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

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

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

--reckless [auto|ephemeral|shared-...]

Obtain a dataset or subdatset and set it up in a potentially unsafe way for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git- annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. Constraints: value must be one of (True, False, 'auto', 'ephemeral') or value must start with 'shared-'

-J NJOBS, --jobs NJOBS

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

--branch *BRANCH*

Clone source at this branch or tag. This option applies only to the top-level dataset not any subdatasets that may be cloned when installing recursively. Note that if the source is a RIA URL with a version, it takes precedence over this option. Constraints: value must be a string 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 no-annex

Synopsis

```
datalad no-annex [-h] [-d DATASET] [--pattern PATTERN [PATTERN ...]] [--ref-dir
    REF_DIR] [--makedirs] [--version]
```

Description

Configure a dataset to never put some content into the dataset's annex

This can be useful in mixed datasets that also contain textual data, such as source code, which can be efficiently and more conveniently managed directly in Git.

Patterns generally look like this:

```
code/*
```

which would match all file in the code directory. In order to match all files under `code/`, including all its subdirectories use such a pattern:

```
code/**
```

Note that this command works incrementally, hence any existing configuration (e.g. from a previous plugin run) is amended, not replaced.

Options

-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 configure. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--pattern *PATTERN* [*PATTERN* ...]

list of path patterns. Any content whose path is matching any pattern will not be annexed when added to a dataset, but instead will be tracked directly in Git. Path pattern have to be relative to the directory given by the REF_DIR option. By default, patterns should be relative to the root of the dataset.

--ref-dir *REF_DIR*

Relative path (within the dataset) to the directory that is to be configured. All patterns are interpreted relative to this path, and configuration is written to a `.gitattributes` file in this directory. [Default: '.']

--makedirs

If set, any missing directories will be created in order to be able to place a file into `--ref-dir`.

--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 remove

Synopsis

```
datalad remove [-h] [-d DATASET] [--drop {datasets|all}] [--reckless
    {modification|availability|undead|kill}] [-m MESSAGE] [-J NJOBS]
    [--recursive] [--nocheck] [--nosave] [--if-dirty IF_DIRTY]
    [--version] [PATH ...]
```

Description

Remove components from datasets

Removing "unlinks" a dataset component, such as a file or subdataset, from a dataset. Such a removal advances the state of a dataset, just like adding new content. A remove operation can be undone, by restoring a previous dataset state, but might require re-obtaining file content and subdatasets from remote locations.

This command relies on the 'drop' command for safe operation. By default, only file content from datasets which will be uninstalled as part of a removal will be dropped. Otherwise file content is retained, such that restoring a previous version also immediately restores file content access, just as it is the case for files directly committed to Git. This default behavior can be changed to always drop content prior removal, for cases where a minimal storage footprint for local datasets installations is desirable.

Removing a dataset component is always a recursive operation. Removing a directory, removes all content underneath the directory too. If subdatasets are located under a to-be-removed path, they will be uninstalled entirely, and all their content dropped. If any subdataset can not be uninstalled safely, the remove operation will fail and halt.

Changed in version 0.16

More in-depth and comprehensive safety-checks are now performed by default. The `--if-dirty` argument is ignored, will be removed in a future release, and can be removed for a safe-by-default behavior. For other cases consider the `--reckless` argument. The `--save` argument is ignored and will be removed in a future release, a dataset modification is now always saved. Consider save's `--amend` argument for post-remove fix-ups. The `--recursive` argument is ignored, and will be removed in a future release. Removal operations are always recursive, and the parameter can be stripped from calls for a safe-by-default behavior.

Deprecated in version 0.16

The `--check` argument will be removed in a future release. It needs to be replaced with `--reckless`.

Examples

Permanently remove a subdataset (and all further subdatasets contained in it) from a dataset:

```
% datalad remove -d <path/to/dataset> <path/to/subds>
```

Permanently remove a superdataset (with all subdatasets) from the filesystem:

```
% datalad remove -d <path/to/dataset>
```

DANGER-ZONE: Fast wipe-out a dataset and all its subdataset, bypassing all safety checks:

```
% datalad remove -d <path/to/dataset> --reckless kill
```

Options

PATH

path of a dataset or dataset component to be removed. Constraints: value must be a string or value must be NONE

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

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

-d DATASET, --dataset DATASET

specify the dataset to perform remove from. If no dataset is given, the current working directory is used as operation context. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--drop {datasets|all}

which dataset components to drop prior removal. This parameter is passed on to the underlying drop operation as its 'what' argument. Constraints: value must be one of ('datasets', 'all') [Default: 'datasets']

--reckless {modification|availability|undead|kill}

disable individual or all data safety measures that would normally prevent potentially irreversible data-loss. With 'modification', unsaved modifications in a dataset will not be detected. This improves performance at the cost of permitting potential loss of unsaved or untracked dataset components. With 'availability', detection of dataset/branch-states that are only available in the local dataset, and detection of an insufficient number of file-content copies will be disabled. Especially the latter is a potentially expensive check which might involve numerous network transactions. With 'undead', detection of whether a to-be-removed local annex is still known to exist in the network of dataset-clones is disabled. This could cause zombie-records of invalid file availability. With 'kill', all safety-checks are disabled. Constraints: value must be one of ('modification', 'availability', 'undead', 'kill')

-m MESSAGE, --message MESSAGE

a description of the state or the changes made to a dataset. Constraints: value must be a string or value must be NONE

-J NJOBS, --jobs NJOBS

how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. Constraints: value must be convertible to type 'int' or value must be NONE or value must be one of ('auto',)

--recursive, -r

DEPRECATED and IGNORED: removal is always a recursive operation.

--nocheck

DEPRECATED: use '–reckless availability'.

--nosave

DEPRECATED and IGNORED; use *save –amend* instead.

--if-dirty *IF_DIRTY*

DEPRECATED and IGNORED: use –reckless instead.

--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 subdatasets

Synopsis

```
datalad subdatasets [-h] [-d DATASET] [--state {present|absent|any}] [--fulfilled
    FULFILLED] [-r] [-R LEVELS] [--contains PATH] [--bottomup]
    [--set-property NAME VALUE] [--delete-property NAME] [--version]
    [PATH ...]
```

Description

Report subdatasets and their properties.

The following properties are reported (if possible) for each matching subdataset record.

"name"
Name of the subdataset in the parent (often identical with the relative path in the parent dataset)

"path"
Absolute path to the subdataset

"parentds"
Absolute path to the parent dataset

"gitshasum"
SHA1 of the subdataset commit recorded in the parent dataset

"state"
Condition of the subdataset: 'absent', 'present'

"gitmodule_url"
URL of the subdataset recorded in the parent

"gitmodule_name"
Name of the subdataset recorded in the parent

"gitmodule_<label>"
Any additional configuration property on record.

Performance note: Property modification, requesting BOTTOMUP reporting order, or a particular numerical *recursion_limit* implies an internal switch to an alternative query implementation for recursive query that is more flexible, but also notably slower (performs one call to Git per dataset versus a single call for all combined).

The following properties for subdatasets are recognized by DataLad (without the 'gitmodule_' prefix that is used in the query results):

"datalad-recursiveinstall"
: If set to 'skip', the respective subdataset is skipped when DataLad is recursively installing its superdataset. However, the subdataset remains installable when explicitly requested, and no other features are impaired.

"datalad-url"
: If a subdataset was originally established by cloning, 'datalad-url' records the URL that was used to do so. This might be different from 'url' if the URL contains datalad specific pieces like any URL of the form "ria+<some protocol>…".

Options

PATH

path/name to query for subdatasets. Defaults to the current directory. Constraints: value must be a string or value must be NONE

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

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

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

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

--state {present|absent|any}

indicate which (sub)datasets to consider: either only locally present, absent, or any of those two kinds. Constraints: value must be one of ('present', 'absent', 'any') [Default: 'any']

--fulfilled *FULFILLED*

DEPRECATED: use –state instead. If given, must be a boolean flag indicating whether to consider either only locally present or absent datasets. By default all subdatasets are considered regardless of their status. Constraints: value must be convertible to type bool or value must be NONE [Default: None(DEPRECATED)]

-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

--contains PATH

limit to the subdatasets containing the given path. If a root path of a subdataset is given, the last considered dataset will be the subdataset itself. This option can be given multiple times, in which case datasets that contain any of the given paths will be considered. Constraints: value must be a string or value must be NONE

--bottomup

whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down.

--set-property NAME VALUE

Name and value of one or more subdataset properties to be set in the parent dataset's .gitmodules file. The property name is case-insensitive, must start with a letter, and consist only of alphanumeric characters. The value can be a Python format() template string wrapped in '<>' (e.g. '<{gitmodule_name}>'). Supported keywords are any item reported in the result properties of this command, plus 'refds_relpath' and 'refds_relname': the relative path of a subdataset with respect to the base dataset of the command call, and, in the latter case, the same string with all directory separators replaced by dashes. This option can be given multiple times. Constraints: value must be a string or value must be NONE

--delete-property NAME

Name of one or more subdataset properties to be removed from the parent dataset's .gitmodules file. This option can be given multiple times. Constraints: value must be a string 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 unlock

Synopsis

```
datalad unlock [-h] [-d DATASET] [-r] [-R LEVELS] [--version] [path ...]
```

Description

Unlock file(s) of a dataset

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

Examples

Unlock a single file:

```
% datalad unlock <path/to/file>
```

Unlock all contents in the dataset:

```
% datalad unlock .
```

Options

path

file(s) to unlock. Constraints: value must be a string or value must be NONE

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

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

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

"specify the dataset to unlock files in. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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>.

Dataset siblings and 3rd-party platform support

datalad siblings

Synopsis

```
datalad siblings [-h] [-d DATASET] [-s NAME] [--url [URL]] [--pushurl PUSHURL] [-D
    DESCRIPTION] [--fetch] [--as-common-datasrc NAME]
    [--publish-depends SIBLINGNAME] [--publish-by-default REFSPEC]
    [--annex-wanted EXPR] [--annex-required EXPR] [--annex-group
    EXPR] [--annex-groupwanted EXPR] [--inherit] [--no-annex-info]
    [-r] [-R LEVELS] [--version]
    [{query|add|remove|configure|enable}]
```

Description

Manage sibling configuration

This command offers four different actions: 'query', 'add', 'remove', 'configure', 'enable'. 'query' is the default action and can be used to obtain information about (all) known siblings. 'add' and 'configure' are highly similar actions, the only difference being that adding a sibling with a name that is already registered will fail, whereas re-configuring a (different) sibling under a known name will not be considered an error. 'enable' can be used to complete access configuration for non-Git sibling (aka git-annex special remotes). Lastly, the 'remove' action allows for the removal (or de-configuration) of a registered sibling.

For each sibling (added, configured, or queried) all known sibling properties are reported. This includes:

"name"
: Name of the sibling

"path"
: Absolute path of the dataset

"url"
For regular siblings at minimum a "fetch" URL, possibly also a "pushurl"

Additionally, any further configuration will also be reported using a key that matches that in the Git configuration.

By default, sibling information is rendered as one line per sibling following this scheme:

```
<dataset_path>: <sibling_name>(<+|->) [<access_specification]
```

where the + and - labels indicate the presence or absence of a remote data annex at a particular remote, and ACCESS_SPECIFICATION contains either a URL and/or a type label for the sibling.

Options

{query|add|remove|configure|enable}

command action selection (see general documentation). Constraints: value must be one of ('query', 'add', 'remove', 'configure', 'enable') [Default: 'query']

-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 configure. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-s NAME, --name NAME

name of the sibling. For addition with path "URLs" and sibling removal this option is mandatory, otherwise the hostname part of a given URL is used as a default. This option can be used to limit 'query' to a specific sibling. Constraints: value must be a string or value must be NONE

--url [*URL*]

the URL of or path to the dataset sibling named by NAME. For recursive operation it is required that a template string for building subdataset sibling URLs is given. List of currently available placeholders: %NAME the name of the dataset, where slashes are replaced by dashes. Constraints: value must be a string or value must be NONE

--pushurl *PUSHURL*

in case the URL cannot be used to publish to the dataset sibling, this option specifies a URL to be used instead. If no *url* is given, PUSHURL serves as *url* as well. Constraints: value must be a string or value must be NONE

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

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

--fetch

fetch the sibling after configuration.

--as-common-datasrc NAME

configure a sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset. The sibling must be a regular Git remote with a configured HTTP(S) URL.

--publish-depends SIBLINGNAME

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

--publish-by-default REFSPEC

add a refspec to be published to this sibling by default if nothing specified. Constraints: value must be a string or value must be NONE

--annex-wanted EXPR

expression to specify 'wanted' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-wanted/ for more information. Constraints: value must be a string or value must be NONE

--annex-required EXPR

expression to specify 'required' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-required/ for more information. Constraints: value must be a string or value must be NONE

--annex-group EXPR

expression to specify a group for the repository. See https://git- annex.branchable.com/git-annex-group/ for more information. Constraints: value must be a string or value must be NONE

--annex-groupwanted EXPR

expression for the groupwanted. Makes sense only if –annex-wanted="groupwanted" and annex-group is given too. See https://git-annex.branchable.com/git-annex- groupwanted/ for more information. Constraints: value must be a string or value must be NONE

--inherit

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

--no-annex-info

Whether to query all information about the annex configurations of siblings. Can be disabled if speed is a concern.

-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 create-sibling

Synopsis

```
datalad create-sibling [-h] [-s [NAME]] [--target-dir PATH] [--target-url URL]
    [--target-pushurl URL] [--dataset DATASET] [-r] [-R LEVELS]
    [--existing MODE] [--shared
    {false|true|umask|group|all|world|everybody|0xxx}] [--group
```

(continues on next page)

(continued from previous page)

```
GROUP] [--ui {false|true|html_filename}] [--as-common-datasrc
NAME] [--publish-by-default REFSPEC] [--publish-depends
SIBLINGNAME] [--annex-wanted EXPR] [--annex-group EXPR]
[--annex-groupwanted EXPR] [--inherit] [--since SINCE]
[--version] [SSHURL]
```

Description

Create a dataset sibling on a UNIX-like Shell (local or SSH)-accessible machine

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

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

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

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

Options

SSHURL

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

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

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

-s [NAME], --name [NAME]

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

--target-dir PATH

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

--target-url URL

"public" access URL of the to-be-created target dataset(s) (default: SSHURL). Accessibility of this URL determines the access permissions of potential consumers of the dataset. As with *target_dir*, templates (same set of placeholders) are supported. Also, if specified, it is provided as the annex description. Constraints: value must be a string or value must be NONE

--target-pushurl URL

In case the TARGET_URL cannot be used to publish to the dataset, this option specifies an alternative URL for this purpose. As with *target_url*, templates (same set of placeholders) are supported. Constraints: value must be a string or value must be NONE

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

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

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

--existing MODE

action to perform, if a sibling is already configured under the given name and/or a target (non-empty) directory already exists. In this case, a dataset can be skipped ('skip'), the sibling configuration be updated ('reconfigure'), or process interrupts with error ('error'). DANGER ZONE: If 'replace' is used, an existing target directory will be forcefully removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). REPLACE could lead to data loss, so use with care. To minimize possibility of data loss, in interactive mode DataLad will ask for confirmation, but it would raise an exception in non- interactive mode. Constraints: value must be one of ('skip', 'error', 'reconfigure', 'replace') [Default: 'error']

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

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

--group GROUP

Filesystem group for the repository. Specifying the group is particularly important when –shared=group. Constraints: value must be a string or value must be NONE

--ui {false|true|html_filename}

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

--as-common-datasrc NAME

configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto- enabled special remote).

--publish-by-default REFSPEC

add a refspec to be published to this sibling by default if nothing specified. Constraints: value must be a string or value must be NONE

--publish-depends SIBLINGNAME

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

--annex-wanted EXPR

expression to specify 'wanted' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-wanted/ for more information. Constraints: value must be a string or value must be NONE

--annex-group EXPR

expression to specify a group for the repository. See https://git- annex.branchable.com/git-annex-group/ for more information. Constraints: value must be a string or value must be NONE

--annex-groupwanted EXPR

expression for the groupwanted. Makes sense only if –annex-wanted="groupwanted" and annex-group is given too. See https://git-annex.branchable.com/git-annex- groupwanted/ for more information. Constraints: value must be a string or value must be NONE

--inherit

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

--since *SINCE*

limit processing to subdatasets that have been changed since a given state (by tag, branch, commit, etc). This can be used to create siblings for recently added subdatasets. If '^' is given, the last state of the current branch at the sibling is taken as a starting point. Constraints: value must be a string 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 create-sibling-github

Synopsis

```
datalad create-sibling-github [-h] [--dataset DATASET] [-r] [-R LEVELS] [-s NAME] [--
↪existing
    {skip|error|reconfigure|replace}] [--github-login TOKEN]
    [--credential NAME] [--github-organization NAME]
    [--access-protocol {https|ssh|https-ssh}] [--publish-depends
    SIBLINGNAME] [--private] [--description DESCRIPTION] [--dryrun]
    [--dry-run] [--api URL] [--version] [<org-name>/]<repo-basename>
```

Description

Create dataset sibling on GitHub.org (or an enterprise deployment).

GitHub is a popular commercial solution for code hosting and collaborative development. GitHub cannot host dataset content (but see LFS, http://handbook.datalad.org/r.html?LFS). However, in combination with other data sources and siblings, publishing a dataset to GitHub can facilitate distribution and exchange, while still allowing any dataset consumer to obtain actual data content from alternative sources.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Settings->Developer Settings->Personal access tokens->Generate new token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Github instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

Changed in version 0.16
The API has been aligned with the some `create-sibling-...` commands of other GitHub-like services, such as GOGS, GIN, GitTea.

Deprecated in version 0.16
The `--dryrun` option will be removed in a future release, use the renamed `--dry-run` option instead. The `--github-login` option will be removed in a future release, use the `--credential` option instead. The `--github-organization` option will be removed in a future release, prefix the reposity name with `<org>/` instead.

Examples

Use a new sibling on GIN as a common data source that is auto- available when cloning from GitHub:

```
% datalad create-sibling-gin myrepo -s gin

# the sibling on GitHub will be used for collaborative work
% datalad create-sibling-github myrepo -s github

# register the storage of the public GIN repo as a data source
% datalad siblings configure -s gin --as-common-datasrc gin-storage

# announce its availability on github
% datalad push --to github
```

Options

[<org-name>/]<repo-(base)name>

repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string

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

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

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

dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

-s NAME, --name NAME

name of the sibling in the local dataset installation (remote name). Constraints: value must be a string or value must be NONE [Default: 'github']

--existing {skip|error|reconfigure|replace}

behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). REPLACE could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. Constraints: value must be one of ('skip', 'error', 'reconfigure', 'replace') [Default: 'error']

--github-login TOKEN

Deprecated, use the credential parameter instead. If given must be a personal access token. Constraints: value must be a string or value must be NONE

--credential NAME

name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). Constraints: value must be a string or value must be NONE

--github-organization NAME

Deprecated, prepend a repo name with an '<orgname>/' prefix instead. Constraints: value must be a string or value must be NONE

--access-protocol {https|ssh|https-ssh}

access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. Constraints: value must be one of ('https', 'ssh', 'https-ssh') [Default: 'https']

--publish-depends SIBLINGNAME

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

--private

if set, create a private repository.

--description *DESCRIPTION*

Brief description, displayed on the project's page. Constraints: value must be a string or value must be NONE

--dryrun

Deprecated. Use the renamed `--dry-run` parameter.

--dry-run

if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets.

--api URL

URL of the GitHub instance API. Constraints: value must be a string or value must be NONE [Default: 'https://api.github.com']

--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 create-sibling-gitlab

Synopsis

```
datalad create-sibling-gitlab [-h] [--site SITENAME] [--project NAME/LOCATION] [--layout
    {collection|flat}] [--dataset DATASET] [-r] [-R LEVELS] [-s
    NAME] [--existing {skip|error|reconfigure}] [--access
    {http|ssh|ssh+http}] [--publish-depends SIBLINGNAME]
    [--description DESCRIPTION] [--dryrun] [--dry-run] [--version]
    [PATH ...]
```

Description

Create dataset sibling at a GitLab site

An existing GitLab project, or a project created via the GitLab web interface can be configured as a sibling with the siblings command. Alternatively, this command can create a GitLab project at any location/path a given user has appropriate permissions for. This is particularly helpful for recursive sibling creation for subdatasets. API access and authentication are implemented via python-gitlab, and all its features are supported. A particular GitLab site must be configured in a named section of a python-gitlab.cfg file (see https://python-gitlab.readthedocs.io/en/stable/cli-usage.html#configuration-file-format for details), such as:

```
[mygit]
url = https://git.example.com
api_version = 4
private_token = abcdefghijklmnopqrst
```

Subsequently, this site is identified by its name ('mygit' in the example above).

(Recursive) sibling creation for all, or a selected subset of subdatasets is supported with two different project layouts (see –layout):

"flat"
: All datasets are placed as GitLab projects in the same group. The project name of the top-level dataset follows the configured datalad.gitlab-SITENAME-project configuration. The project names of contained subdatasets extend the configured name with the subdatasets' s relative path within the root dataset, with all path separator characters replaced by '-'. This path separator is configurable (see Configuration).

"collection"
A new group is created for the dataset hierarchy, following the datalad.gitlab-SITENAME-project configuration. The root dataset is placed in a "project" project inside this group, and all nested subdatasets are represented inside the group using a "flat" layout. The root datasets project name is configurable (see Configuration).

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

Configuration

Many configuration switches and options for GitLab sibling creation can be provided as arguments to the command. However, it is also possible to specify a particular setup in a dataset's configuration. This is particularly important when managing large collections of datasets. Configuration options are:

"datalad.gitlab-default-site"
Name of the default GitLab site (see –site)

"datalad.gitlab-SITENAME-siblingname"
Name of the sibling configured for the local dataset that points to the GitLab instance SITENAME (see –name)

"datalad.gitlab-SITENAME-layout"
Project layout used at the GitLab instance SITENAME (see –layout)

"datalad.gitlab-SITENAME-access"
Access method used for the GitLab instance SITENAME (see –access)

"datalad.gitlab-SITENAME-project"
Project "location/path" used for a datasets at GitLab instance SITENAME (see –project). Configuring this is useful for deriving project paths for subdatasets, relative to superdataset. The root-level group ("location") needs to be created beforehand via GitLab's web interface.

"datalad.gitlab-default-projectname"
The collection layout publishes (sub)datasets as projects with a custom name. The default name "project" can be overridden with this configuration.

"datalad.gitlab-default-pathseparator"
The flat and collection layout represent subdatasets with project names that correspond to their path within the superdataset, with the regular path separator replaced with a "-": superdataset-subdataset. This configuration can be used to override this default separator.

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gitlab instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

Options

PATH

selectively create siblings for any datasets underneath a given path. By default only the root dataset is considered.

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

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

--site SITENAME

name of the GitLab site to create a sibling at. Must match an existing python- gitlab configuration section with location and authentication settings (see https://python-gitlab.readthedocs.io/en/stable/cli-usage.html#configuration). By default the dataset configuration is consulted. Constraints: value must be NONE or value must be a string

--project NAME/LOCATION

project name/location at the GitLab site. If a subdataset of the reference dataset is processed, its project path is automatically determined by the LAYOUT configuration, by default. Users need to create the root-level GitLab group (NAME) via the webinterface before running the command. Constraints: value must be NONE or value must be a string

--layout {collection|flat}

layout of projects at the GitLab site, if a collection, or a hierarchy of datasets and subdatasets is to be created. By default the dataset configuration is consulted. Constraints: value must be one of ('collection', 'flat')

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

reference or root dataset. If no path constraints are given, a sibling for this dataset will be created. In this and all other cases, the reference dataset is also consulted for the GitLab configuration, and desired project layout. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

-s NAME, --name NAME

name to represent the GitLab sibling remote in the local dataset installation. If not specified a name is looked up in the dataset configuration, or defaults to the SITE name. Constraints: value must be a string or value must be NONE

--existing {skip|error|reconfigure}

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

--access {http|ssh|ssh+http}

access method used for data transfer to and from the sibling. 'ssh': read and write access used the SSH protocol; 'http': read and write access use HTTP requests; 'ssh+http': read access is done via HTTP and write access performed with SSH. Dataset configuration is consulted for a default, 'http' is used otherwise. Constraints: value must be one of ('http', 'ssh', 'ssh+http')

--publish-depends SIBLINGNAME

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

--description *DESCRIPTION*

brief description for the GitLab project (displayed on the site). Constraints: value must be a string or value must be NONE

--dryrun

Deprecated. Use the renamed `--dry-run` parameter.

--dry-run

if set, no repository will be created, only tests for name collisions will be performed, and would-be repository names are reported for all relevant datasets.

--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 create-sibling-gogs

Synopsis

```
datalad create-sibling-gogs [-h] [--api URL] [--dataset DATASET] [-r] [-R LEVELS] [-s␣
↪NAME]
    [--existing {skip|error|reconfigure|replace}] [--credential
    NAME] [--access-protocol {https|ssh|https-ssh}]
    [--publish-depends SIBLINGNAME] [--private] [--description
    DESCRIPTION] [--dry-run] [--version]
    [<org-name>/]<repo-basename>
```

Description

Create a dataset sibling on a GOGS site

GOGS is a self-hosted, free and open source code hosting solution with low resource demands that enable running it on inexpensive devices like a Raspberry Pi, or even directly on a NAS device.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Your Settings->Applications->Generate New Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gogs instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16

Options

[<org-name>/]<repo-(base)name>

repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string

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

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

--api URL

URL of the GOGS instance without a 'api/<version>' suffix. Constraints: value must be a string or value must be NONE

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

dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

-s NAME, --name NAME

name of the sibling in the local dataset installation (remote name). Constraints: value must be a string or value must be NONE

--existing {skip|error|reconfigure|replace}

behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). REPLACE could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. Constraints: value must be one of ('skip', 'error', 'reconfigure', 'replace') [Default: 'error']

--credential NAME

name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). Constraints: value must be a string or value must be NONE

--access-protocol {https|ssh|https-ssh}

access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. Constraints: value must be one of ('https', 'ssh', 'https-ssh') [Default: 'https']

--publish-depends SIBLINGNAME

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

--private

if set, create a private repository.

--description *DESCRIPTION*

Brief description, displayed on the project's page. Constraints: value must be a string or value must be NONE

--dry-run

if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets.

--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 create-sibling-gitea

Synopsis

```
datalad create-sibling-gitea [-h] [--dataset DATASET] [-r] [-R LEVELS] [-s NAME] [--
↪existing
    {skip|error|reconfigure|replace}] [--api URL] [--credential
    NAME] [--access-protocol {https|ssh|https-ssh}]
    [--publish-depends SIBLINGNAME] [--private] [--description
    DESCRIPTION] [--dry-run] [--version]
    [<org-name>/]<repo-basename>
```

Description

Create a dataset sibling on a Gitea site

Gitea is a lightweight, free and open source code hosting solution with low resource demands that enable running it on inexpensive devices like a Raspberry Pi.

This command uses the main Gitea instance at https://gitea.com as the default target, but other deployments can be used via the 'api' parameter.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Settings->Applications->Generate Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gitea instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16

Options

[<org-name>/]<repo-(base)name>

repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string

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

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

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

dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

-s NAME, --name NAME

name of the sibling in the local dataset installation (remote name). Constraints: value must be a string or value must be NONE [Default: 'gitea']

--existing {skip|error|reconfigure|replace}

behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). REPLACE could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. Constraints: value must be one of ('skip', 'error', 'reconfigure', 'replace') [Default: 'error']

--api URL

URL of the Gitea instance without a 'api/<version>' suffix. Constraints: value must be a string or value must be NONE [Default: 'https://gitea.com']

--credential NAME

name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). Constraints: value must be a string or value must be NONE

--access-protocol {https|ssh|https-ssh}

access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. Constraints: value must be one of ('https', 'ssh', 'https-ssh') [Default: 'https']

--publish-depends SIBLINGNAME

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

--private

if set, create a private repository.

--description *DESCRIPTION*

Brief description, displayed on the project's page. Constraints: value must be a string or value must be NONE

--dry-run

if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets.

--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 create-sibling-gin

Synopsis

```
datalad create-sibling-gin [-h] [--dataset DATASET] [-r] [-R LEVELS] [-s NAME] [--
↪existing
    {skip|error|reconfigure|replace}] [--api URL] [--credential
    NAME] [--access-protocol {https|ssh|https-ssh}]
    [--publish-depends SIBLINGNAME] [--private] [--description
    DESCRIPTION] [--dry-run] [--version]
    [<org-name>/]<repo-basename>
```

Description

Create a dataset sibling on a GIN site (with content hosting)

GIN (G-Node infrastructure) is a free data management system. It is a GitHub-like, web-based repository store and provides fine-grained access control to shared data. GIN is built on Git and git-annex, and can natively host DataLad datasets, including their data content!

This command uses the main GIN instance at https://gin.g-node.org as the default target, but other deployments can be used via the 'api' parameter.

An SSH key, properly registered at the GIN instance, is required for data upload via DataLad. Data download from public projects is also possible via anonymous HTTP.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Your Settings->Applications->Generate New Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gin instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16

Examples

Create a repo 'myrepo' on GIN and register it as sibling 'mygin':

```
% datalad create-sibling-gin myrepo -s mygin
```

Create private repos with name(-prefix) 'myrepo' on GIN for a dataset and all its present subdatasets:

```
% datalad create-sibling-gin myrepo -r --private
```

Create a sibling repo on GIN, and register it as a common data source in the dataset that is available regardless of whether the dataset was directly cloned from GIN:

```
% datalad create-sibling-gin myrepo -s gin
# first push creates git-annex branch remotely and obtains annex UUID
% datalad push --to gin
% datalad siblings configure -s gin --as-common-datasrc gin-storage
# announce availability (redo for other siblings)
% datalad push --to gin
```

Options

[<org-name>/]<repo-(base)name>

repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset. Constraints: value must be a string

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

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

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

dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

-s NAME, --name NAME

name of the sibling in the local dataset installation (remote name). Constraints: value must be a string or value must be NONE [Default: 'gin']

--existing {skip|error|reconfigure|replace}

behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). REPLACE could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. Constraints: value must be one of ('skip', 'error', 'reconfigure', 'replace') [Default: 'error']

--api URL

URL of the GIN instance without an 'api/<version>' suffix. Constraints: value must be a string or value must be NONE [Default: 'https://gin.g-node.org']

--credential NAME

name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). Constraints: value must be a string or value must be NONE

--access-protocol {https|ssh|https-ssh}

access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. Constraints: value must be one of ('https', 'ssh', 'https-ssh') [Default: 'https-ssh']

--publish-depends SIBLINGNAME

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

--private

if set, create a private repository.

--description *DESCRIPTION*

Brief description, displayed on the project's page. Constraints: value must be a string or value must be NONE

--dry-run

if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets.

--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 create-sibling-ria

Synopsis

```
datalad create-sibling-ria [-h] -s NAME [-d DATASET] [--storage-name NAME] [--alias␣
↪ALIAS]
    [--post-update-hook] [--shared
    {false|true|umask|group|all|world|everybody|0xxx}] [--group
    GROUP] [--storage-sibling MODE] [--existing MODE]
    [--new-store-ok] [--trust-level TRUST-LEVEL] [-r] [-R LEVELS]
    [--no-storage-sibling] [--push-url
    ria+<ssh|file>://<host>[/path]] [--version]
    ria+<ssh|file|https>://<host>[/path]
```

Description

Creates a sibling to a dataset in a RIA store

Communication with a dataset in a RIA store is implemented via two siblings. A regular Git remote (repository sibling) and a git-annex special remote for data transfer (storage sibling) – with the former having a publication dependency on the latter. By default, the name of the storage sibling is derived from the repository sibling's name by appending "-storage".

The store's base path is expected to not exist, be an empty directory, or a valid RIA store.

Notes

RIA URL format

Interactions with new or existing RIA stores require RIA URLs to identify the store or specific datasets inside of it.

The general structure of a RIA URL pointing to a store takes the form `ria+[scheme]://<storelocation>` (e.g., `ria+ssh://[user@]hostname:/absolute/path/to/ria-store`, or `ria+file:///absolute/path/to/ria-store`)

The general structure of a RIA URL pointing to a dataset in a store (for example for cloning) takes a similar form, but appends either the datasets UUID or a "~" symbol followed by the dataset's alias name: `ria+[scheme]://<storelocation>#<dataset-UUID>` or `ria+[scheme]://<storelocation>#~<aliasname>`. In addition, specific version identifiers can be appended to the URL with an additional "@" symbol: `ria+[scheme]://<storelocation>#<dataset-UUID>@<dataset-version>`, where `dataset-version` refers to a branch or tag.

RIA store layout

A RIA store is a directory tree with a dedicated subdirectory for each dataset in the store. The subdirectory name is constructed from the DataLad dataset ID, e.g. `124/68afe-59ec-11ea-93d7-f0d5bf7b5561`, where the first three characters of the ID are used for an intermediate subdirectory in order to mitigate files system limitations for stores containing a large number of datasets.

By default, a dataset in a RIA store consists of two components: A Git repository (for all dataset contents stored in Git) and a storage sibling (for dataset content stored in git-annex).

It is possible to selectively disable either component using `storage-sibling 'off'` or `storage-sibling 'only'`, respectively. If neither component is disabled, a dataset's subdirectory layout in a RIA store contains a standard bare Git repository and an `annex/` subdirectory inside of it. The latter holds a Git-annex object store and comprises the

storage sibling. Disabling the standard git-remote (`storage-sibling='only'`) will result in not having the bare git repository, disabling the storage sibling (`storage-sibling='off'`) will result in not having the `annex/` subdirectory.

Optionally, there can be a further subdirectory `archives` with (compressed) 7z archives of annex objects. The storage remote is able to pull annex objects from these archives, if it cannot find in the regular annex object store. This feature can be useful for storing large collections of rarely changing data on systems that limit the number of files that can be stored.

Each dataset directory also contains a `ria-layout-version` file that identifies the data organization (as, for example, described above).

Lastly, there is a global `ria-layout-version` file at the store's base path that identifies where dataset subdirectories themselves are located. At present, this file must contain a single line stating the version (currently "1"). This line MUST end with a newline character.

It is possible to define an alias for an individual dataset in a store by placing a symlink to the dataset location into an `alias/` directory in the root of the store. This enables dataset access via URLs of format: `ria+<protocol>://<storelocation>#~<aliasname>`.

Compared to standard git-annex object stores, the `annex/` subdirectories used as storage siblings follow a different layout naming scheme ('dirhashmixed' instead of 'dirhashlower'). This is mostly noted as a technical detail, but also serves to remind git-annex powerusers to refrain from running git-annex commands directly in-store as it can cause severe damage due to the layout difference. Interactions should be handled via the ORA special remote instead.

Error logging

To enable error logging at the remote end, append a pipe symbol and an "l" to the version number in ria-layout-version (like so: `1|l\n`).

Error logging will create files in an "error_log" directory whenever the git-annex special remote (storage sibling) raises an exception, storing the Python traceback of it. The logfiles are named according to the scheme `<dataset id>.<annex uuid of the remote>.log` showing "who" ran into this issue with which dataset. Because logging can potentially leak personal data (like local file paths for example), it can be disabled client-side by setting the configuration variable `annex.ora-remote.<storage-sibling-name>.ignore-remote-config`.

Options

ria+<ssh|file|http(s)>://<host>[/path]

URL identifying the target RIA store and access protocol. If `--push-url` is given in addition, this is used for read access only. Otherwise it will be used for write access too and to create the repository sibling in the RIA store. Note, that HTTP(S) currently is valid for consumption only thus requiring to provide `--push-url`. Constraints: value must be a string or value must be NONE

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

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

-s NAME, --name NAME

Name of the sibling. With RECURSIVE, the same name will be used to label all the subdatasets' siblings. Constraints: value must be a string or value must be NONE

-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. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--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. Constraints: value must be a string or value must be NONE

--alias ALIAS

Alias for the dataset in the RIA store. Add the necessary symlink so that this dataset can be cloned from the RIA store using the given ALIAS instead of its ID. With *recursive=True*, only the top dataset will be aliased. Constraints: value must be a string or value must be NONE

--post-update-hook

Enable Git's default post-update-hook for the created sibling. This is useful when the sibling is made accessible via a "dumb server" that requires running 'git update-server-info' to let Git interact properly with it.

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

If given, configures the permissions in the RIA store for multi-users access. Possible values for this option are identical to those of *git init –shared* and are described in its documentation. Constraints: value must be a string or value must be convertible to type bool or value must be NONE

--group GROUP

Filesystem group for the repository. Specifying the group is crucial when –shared=group. Constraints: value must be a string or value must be NONE

--storage-sibling MODE

By default, an ORA storage sibling and a Git repository sibling are created (on). Alternatively, creation of the storage sibling can be disabled (off), or a storage sibling created only and no Git sibling (only). In the latter mode, no Git installation is required on the target host. Constraints: value must be one of ('only',) or value must be convertible to type bool or value must be NONE [Default: True]

--existing MODE

Action to perform, if a (storage) sibling is already configured under the given name and/or a target already exists. In this case, a dataset can be skipped ('skip'), an existing target repository be forcefully re-initialized, and the sibling (re-)configured ('reconfigure'), or the command be instructed to fail ('error'). Constraints: value must be one of ('skip', 'error', 'reconfigure') [Default: 'error']

--new-store-ok

When set, a new store will be created, if necessary. Otherwise, a sibling will only be created if the url points to an existing RIA store.

--trust-level TRUST-LEVEL

specify a trust level for the storage sibling. If not specified, the default git-annex trust level is used. 'trust' should be used with care (see the git- annex-trust man page). Constraints: value must be one of ('trust', 'semitrust', 'untrust')

-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

--no-storage-sibling

This option is deprecated. Use '–storage-sibling off' instead.

--push-url ria+<ssh|file>://<host>[/path]

URL identifying the target RIA store and access protocol for write access to the storage sibling. If given this will also be used for creation of the repository sibling in the RIA store. Constraints: value must be a string 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 export-archive

Synopsis

```
datalad export-archive [-h] [-d DATASET] [-t {tar|zip}] [-c {gz|bz2|}] [--missing-content
    {error|continue|ignore}] [--version] [PATH]
```

Description

Export the content of a dataset as a TAR/ZIP archive.

Options

PATH

File name of the generated TAR archive. If no file name is given the archive will be generated in the current directory and will be named: datalad_<dataset_uuid>.(tar.*|zip). To generate that file in a different directory, provide an existing directory as the file name. Constraints: value must be a string or value must be NONE

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

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

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

"specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-t {tar|zip}, --archivetype {tar|zip}

Type of archive to generate. Constraints: value must be one of ('tar', 'zip') [Default: 'tar']

-c {gz|bz2|}, --compression {gz|bz2|}

Compression method to use. 'bz2' is not supported for ZIP archives. No compression is used when an empty string is given. Constraints: value must be one of ('gz', 'bz2', '') [Default: 'gz']

--missing-content {error|continue|ignore}

By default, any discovered file with missing content will result in an error and the export is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. Constraints: value must be one of ('error', 'continue', 'ignore') [Default: 'error']

--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 export-archive-ora

Synopsis

```
datalad export-archive-ora [-h] [-d DATASET] [--for LABEL] [--annex-wanted FILTERS] [--
↪from FROM
    [FROM ...]] [--missing-content {error|continue|ignore}]
    [--version] TARGET ...
```

Description

Export an archive of a local annex object store for the ORA remote.

Keys in the local annex object store are reorganized in a temporary directory (using links to avoid storage duplication) to use the 'hashdirlower' setup used by git-annex for bare repositories and the directory-type special remote. This alternative object store is then moved into a 7zip archive that is suitable for use in a ORA remote dataset store. Placing such an archive into:

```
<dataset location>/archives/archive.7z
```

Enables the ORA special remote to locate and retrieve all keys contained in the archive.

Options

TARGET

if an existing directory, an 'archive.7z' is placed into it, otherwise this is the path to the target archive. Constraints: value must be a string or value must be NONE

...

list of options for 7z to replace the default '-mx0' to generate an uncompressed archive.

-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. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--for LABEL

name of the target sibling, wanted/preferred settings will be used to filter the files added to the archives. Constraints: value must be a string or value must be NONE

--annex-wanted FILTERS

git-annex-preferred-content expression for git-annex find to filter files. Should start with 'or' or 'and' when used in combination with *–for*.

--from FROM [FROM ...]

one or multiple tree-ish from which to select files.

--missing-content {error|continue|ignore}

By default, any discovered file with missing content will result in an error and the export is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. Constraints: value must be one of ('error', 'continue', 'ignore') [Default: 'error']

--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 export-to-figshare

Synopsis

```
datalad export-to-figshare [-h] [-d DATASET] [--missing-content {error|continue|ignore}]
    [--no-annex] [--article-id ID] [--version] [PATH]
```

Description

Export the content of a dataset as a ZIP archive to figshare

Very quick and dirty approach. Ideally figshare should be supported as a proper git annex special remote. Unfortunately, figshare does not support having directories, and can store only a flat list of files. That makes it impossible for any sensible publishing of complete datasets.

The only workaround is to publish dataset as a zip-ball, where the entire content is wrapped into a .zip archive for which figshare would provide a navigator.

Options

PATH

File name of the generated ZIP archive. If no file name is given the archive will be generated in the top directory of the dataset and will be named: datalad_<dataset_uuid>.zip. Constraints: value must be a string or value must be NONE

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

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

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

"specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--missing-content {error|continue|ignore}

By default, any discovered file with missing content will result in an error and the plugin is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. Constraints: value must be one of ('error', 'continue', 'ignore') [Default: 'error']

--no-annex

By default the generated .zip file would be added to annex, and all files would get registered in git-annex to be available from such a tarball. Also upon upload we will register for that archive to be a possible source for it in annex. Setting this flag disables this behavior.

--article-id ID

Which article to publish to. 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 update

Synopsis

```
datalad update [-h] [-s SIBLING] [--merge [ALLOWED]] [--how
    [{fetch|merge|ff-only|reset|checkout}]] [--how-subds
    [{fetch|merge|ff-only|reset|checkout}]] [--follow
    {sibling|parentds|parentds-lazy}] [-d DATASET] [-r] [-R LEVELS]
    [--fetch-all] [--reobtain-data] [--version] [PATH ...]
```

Description

Update a dataset from a sibling.

Examples

Update from a particular sibling:

```
% datalad update -s <siblingname>
```

Update from a particular sibling and merge the changes from a configured or matching branch from the sibling (see –follow for details):

```
% datalad update --how=merge -s <siblingname>
```

Update from the sibling 'origin', traversing into subdatasets. For subdatasets, merge the revision registered in the parent dataset into the current branch:

```
% datalad update -s origin --how=merge --follow=parentds -r
```

Fetch and merge the remote tracking branch into the current dataset. Then update each subdataset by resetting its current branch to the revision registered in the parent dataset, fetching only if the revision isn't already present:

```
% datalad update --how=merge --how-subds=reset --follow=parentds-lazy -r
```

Options

PATH

constrain to-be-updated subdatasets to the given path for recursive operation. Constraints: value must be a string or value must be NONE

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

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

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

name of the sibling to update from. When unspecified, updates from all siblings are fetched. If there is more than one sibling and changes will be brought into the working tree (as requested via –merge, –how, or –how-subds), a sibling will be chosen based on the configured remote for the current branch. Constraints: value must be a string or value must be NONE

--merge [ALLOWED]

merge obtained changes from the sibling. This is a subset of the functionality that can be achieved via the newer –how. –merge or –merge=any is equivalent to –how=merge. –merge=ff-only is equivalent to –how=ff-only. Constraints: value must be convertible to type bool or value must be one of ('any', 'ff- only') [Default: False]

--how [{fetch|merge|ff-only|reset|checkout}]

how to update the dataset. The default ("fetch") simply fetches the changes from the sibling but doesn't incorporate them into the working tree. A value of "merge" or "ff-only" merges in changes, with the latter restricting the allowed merges to fast-forwards. "reset" incorporates the changes with 'git reset –hard <target>', staying on the current branch but discarding any changes that aren't shared with the target. "checkout", on the other hand, runs 'git checkout <target>', switching from the current branch to a detached state. When –recursive is specified, this action will also apply to subdatasets unless overridden by –how-subds. Constraints: value must be one of ('fetch', 'merge', 'ff-only', 'reset', 'checkout')

--how-subds [{fetch|merge|ff-only|reset|checkout}]

Override the behavior of –how in subdatasets. Constraints: value must be one of ('fetch', 'merge', 'ff-only', 'reset', 'checkout')

--follow {sibling|parentds|parentds-lazy}

source of updates for subdatasets. For 'sibling', the update will be done by merging in a branch from the (specified or inferred) sibling. The branch brought in will either be the current branch's configured branch, if it points to a branch that belongs to the sibling, or a sibling branch with a name that matches the current branch. For 'parentds', the revision registered in the parent dataset of the subdataset is merged in. 'parentds-lazy' is like 'parentds', but prevents fetching from a subdataset's sibling if the registered revision is present in the subdataset. Note that the current dataset is always updated according to 'sibling'. This option has no effect unless a merge is requested and –recursive is specified. Constraints: value must be one of ('sibling', 'parentds', 'parentds-lazy') [Default: 'sibling']

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

specify the dataset to update. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-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

--fetch-all

this option has no effect and will be removed in a future version. When no siblings are given, an all-sibling update will be performed.

--reobtain-data

if enabled, file content that was present before an update will be re-obtained in case a file was changed by the update.

--version

show the module and its version which provides the command

Authors

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

Reproducible execution

Extending the functionality of the core `run` command.

datalad rerun

Synopsis

```
datalad rerun [-h] [--since SINCE] [-d DATASET] [-b NAME] [-m MESSAGE] [--onto base]
    [--script FILE] [--report] [--assume-ready
    {inputs|outputs|both}] [--explicit] [-J NJOBS] [--version]
    [REVISION]
```

Description

Re-execute previous *datalad run* commands.

This will unlock any dataset content that is on record to have been modified by the command in the specified revision. It will then re-execute the command in the recorded path (if it was inside the dataset). Afterwards, all modifications will be saved.

Report mode

When called with –report, this command reports information about what would be re-executed as a series of records. There will be a record for each revision in the specified revision range. Each of these will have one of the following "rerun_action" values:

- run: the revision has a recorded command that would be re-executed
- skip-or-pick: the revision does not have a recorded command and would be either skipped or cherry picked
- merge: the revision is a merge commit and a corresponding merge would be made

The decision to skip rather than cherry pick a revision is based on whether the revision would be reachable from HEAD at the time of execution.

In addition, when a starting point other than HEAD is specified, there is a rerun_action value "checkout", in which case the record includes information about the revision the would be checked out before rerunning any commands.

NOTE
Currently the "onto" feature only sets the working tree of the current dataset to a previous state. The working trees of any subdatasets remain unchanged.

Examples

Re-execute the command from the previous commit:

```
% datalad rerun
```

Re-execute any commands in the last five commits:

```
% datalad rerun --since=HEAD~5
```

Do the same as above, but re-execute the commands on top of HEAD~5 in a detached state:

```
% datalad rerun --onto= --since=HEAD~5
```

Re-execute all previous commands and compare the old and new results:

```
% # on master branch
% datalad rerun --branch=verify --since=
% # now on verify branch
% datalad diff --revision=master..
% git log --oneline --left-right --cherry-pick master...
```

Options

REVISION

rerun command(s) in REVISION. By default, the command from this commit will be executed, but –since can be used to construct a revision range. The default value is like "HEAD" but resolves to the main branch when on an adjusted branch. Constraints: value must be a string or value must be NONE

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

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

--since *SINCE*

If SINCE is a commit-ish, the commands from all commits that are reachable from *revision* but not SINCE will be re-executed (in other words, the commands in git log SINCE..REVISION). If SINCE is an empty string, it is set to the parent of the first commit that contains a recorded command (i.e., all commands in git log REVISION will be re-executed). Constraints: value must be a string or value must be NONE

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

specify the dataset from which to rerun a recorded command. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. If a dataset is given, the command will be executed in the root directory of this dataset. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-b NAME, --branch NAME

create and checkout this branch before rerunning the commands. Constraints: value must be a string or value must be NONE

-m MESSAGE, --message MESSAGE

use MESSAGE for the reran commit rather than the recorded commit message. In the case of a multi-commit rerun, all the reran commits will have this message. Constraints: value must be a string or value must be NONE

--onto base

start point for rerunning the commands. If not specified, commands are executed at HEAD. This option can be used to specify an alternative start point, which will be checked out with the branch name specified by –branch or in a detached state otherwise. As a special case, an empty value for this option means the parent of the first run commit in the specified revision list. Constraints: value must be a string or value must be NONE

--script FILE

extract the commands into FILE rather than rerunning. Use - to write to stdout instead. This option implies –report. Constraints: value must be a string or value must be NONE

--report

Don't actually re-execute anything, just display what would be done. Note: If you give this option, you most likely want to set –output-format to 'json' or 'json_pp'.

--assume-ready {inputs|outputs|both}

Assume that inputs do not need to be retrieved and/or outputs do not need to unlocked or removed before running the command. This option allows you to avoid the expense of these preparation steps if you know that they are unnecessary. Note that this option also affects any additional outputs that are automatically inferred based on inspecting changed files in the run commit. Constraints: value must be one of ('inputs', 'outputs', 'both')

--explicit

Consider the specification of inputs and outputs in the run record to be explicit. Don't warn if the repository is dirty, and only save modifications to the outputs from the original record. Note that when several run commits are specified, this applies to every one. Care should also be taken when using –onto because checking out a new HEAD can easily fail when the working tree has modifications.

-J NJOBS, --jobs NJOBS

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

--version

show the module and its version which provides the command

Authors

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

datalad run-procedure

Synopsis

```
datalad run-procedure [-h] [-d PATH] [--discover] [--help-proc] [--version] ...
```

Description

Run prepared procedures (DataLad scripts) on a dataset

Concept

A "procedure" is an algorithm with the purpose to process a dataset in a particular way. Procedures can be useful in a wide range of scenarios, like adjusting dataset configuration in a uniform fashion, populating a dataset with particular content, or automating other routine tasks, such as synchronizing dataset content with certain siblings.

Implementations of some procedures are shipped together with DataLad, but additional procedures can be provided by 1) any DataLad extension, 2) any (sub-)dataset, 3) a local user, or 4) a local system administrator. DataLad will look for procedures in the following locations and order:

Directories identified by the configuration settings

- 'datalad.locations.user-procedures' (determined by platformdirs.user_config_dir; defaults to '$HOME/.config/datalad/procedures' on GNU/Linux systems)
- 'datalad.locations.system-procedures' (determined by platformdirs.site_config_dir; defaults to '/etc/xdg/datalad/procedures' on GNU/Linux systems)
- 'datalad.locations.dataset-procedures'

and subsequently in the 'resources/procedures/' directories of any installed extension, and, lastly, of the DataLad installation itself.

Please note that a dataset that defines 'datalad.locations.dataset-procedures' provides its procedures to any dataset it is a subdataset of. That way you can have a collection of such procedures in a dedicated dataset and install it as a subdataset into any dataset you want to use those procedures with. In case of a naming conflict with such a dataset hierarchy, the dataset you're calling run-procedures on will take precedence over its subdatasets and so on.

Each configuration setting can occur multiple times to indicate multiple directories to be searched. If a procedure matching a given name is found (filename without a possible extension), the search is aborted and this implementation will be executed. This makes it possible for individual datasets, users, or machines to override externally provided procedures (enabling the implementation of customizable processing "hooks").

Procedure implementation

A procedure can be any executable. Executables must have the appropriate permissions and, in the case of a script, must contain an appropriate "shebang" line. If a procedure is not executable, but its filename ends with '.py', it is automatically executed by the 'python' interpreter (whichever version is available in the present environment). Likewise, procedure implementations ending on '.sh' are executed via 'bash'.

Procedures can implement any argument handling, but must be capable of taking at least one positional argument (the absolute path to the dataset they shall operate on).

For further customization there are two configuration settings per procedure available:

- 'datalad.procedures.<NAME>.call-format' fully customizable format string to determine how to execute procedure NAME (see also datalad-run). It currently requires to include the following placeholders:
  - '{script}': will be replaced by the path to the procedure
  - '{ds}': will be replaced by the absolute path to the dataset the procedure shall operate on
  - '{args}': (not actually required) will be replaced by all additional arguments passed into run-procedure after NAME

    As an example the default format string for a call to a python script is: "python {script} {ds} {args}"
- 'datalad.procedures.<NAME>.help' will be shown on *datalad run-procedure –help-proc NAME* to provide a description and/or usage info for procedure NAME

Examples

Find out which procedures are available on the current system:

```
% datalad run-procedure --discover
```

Run the 'yoda' procedure in the current dataset:

```
% datalad run-procedure cfg_yoda
```

Options

NAME [ARGS]

Name and possibly additional arguments of the to-be-executed procedure. [PY: Can also be a dictionary coming from run-procedure(discover=True).]Note, that all options to run-procedure need to be put before NAME, since all ARGS get assigned to NAME.

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

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

-d PATH, --dataset PATH

specify the dataset to run the procedure on. An attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--discover

if given, all configured paths are searched for procedures and one result record per discovered procedure is yielded, but no procedure is executed.

--help-proc

if given, get a help message for procedure NAME from config setting datalad.procedures.NAME.help.

--version

show the module and its version which provides the command

Authors

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

Helpers and support utilities

datalad add-archive-content

Synopsis

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

Description

Add content of an archive under git annex control.

Given an already annex'ed archive, extract and add its files to the dataset, and reference the original archive as a custom special remote.

Examples

Add files from the archive 'big_tarball.tar.gz', but keep big_tarball.tar.gz in the index:

```
% datalad add-archive-content big_tarball.tar.gz
```

Add files from the archive 'tarball.tar.gz', and remove big_tarball.tar.gz from the index:

```
% datalad add-archive-content big_tarball.tar.gz --delete
```

Add files from the archive 's3.zip' but remove the leading directory:

```
% datalad add-archive-content s3.zip --strip-leading-dirs
```

Options

archive

archive file or a key (if –key specified). Constraints: value must be a 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*

"specify the dataset to save. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

--annex *ANNEX*

DEPRECATED. Use the 'dataset' parameter instead.

--add-archive-leading-dir

place extracted content under a directory which would correspond to the archive name with all suffixes stripped. E.g. the content of *archive.tar.gz* will be extracted under *archive/*.

--strip-leading-dirs

remove one or more leading directories from the archive layout on extraction.

--leading-dirs-depth *LEADING_DIRS_DEPTH*

maximum depth of leading directories to strip. If not specified (None), no limit.

--leading-dirs-consider *LEADING_DIRS_CONSIDER*

regular expression(s) for directories to consider to strip away. Constraints: value must be a string or value must be NONE

--use-current-dir

extract the archive under the current directory, not the directory where the archive is located. This parameter is applied automatically if –key was used.

-D, --delete

delete original archive from the filesystem/Git in current tree. Note that it will be of no effect if –key is given.

--key

signal if provided archive is not actually a filename on its own but an annex key. The archive will be extracted in the current directory.

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

regular expressions for filenames which to exclude from being added to annex. Applied after –rename if that one is specified. For exact matching, use anchoring. Constraints: value must be a string or value must be NONE

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

regular expressions to rename files before added them under to Git. The first defines how to split provided string into two parts: Python regular expression (with groups), and replacement string. Constraints: value must be a string or value must be NONE

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

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

-o *ANNEX_OPTIONS*, --annex-options *ANNEX_OPTIONS*

additional options to pass to git-annex. Constraints: value must be a string or value must be NONE

--copy

copy the content of the archive instead of moving.

--no-commit

don't commit upon completion.

--allow-dirty

flag that operating on a dirty repository (uncommitted or untracked content) is ok.

--stats *STATS*

ActivityStats instance for global tracking.

--drop-after

drop extracted files after adding to annex.

--delete-after

extract under a temporary directory, git-annex add, and delete afterwards. To be used to "index" files within annex without actually creating corresponding files under git. Note that *annex dropunused* would later remove that load.

--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 clean

Synopsis

```
datalad clean [-h] [-d DATASET] [--what [WHAT ...]] [--dry-run] [-r] [-R LEVELS]
    [--version]
```

Description

Clean up after DataLad (possible temporary files etc.)

Removes temporary files and directories left behind by DataLad and git-annex in a dataset.

Examples

Clean all known temporary locations of a dataset:

```
% datalad clean
```

Report on all existing temporary locations of a dataset:

```
% datalad clean --dry-run
```

Clean all known temporary locations of a dataset and all its subdatasets:

```
% datalad clean -r
```

Clean only the archive extraction caches of a dataset and all its subdatasets:

```
% datalad clean --what cached-archives -r
```

Report on existing annex transfer files of a dataset and all its subdatasets:

```
% datalad clean --what annex-transfer -r --dry-run
```

Options

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

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

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

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

--what [*WHAT* ...]

What to clean. If none specified – all known targets are considered. Constraints: value must be one of ('cached-archives', 'annex-tmp', 'annex- transfer', 'search-index') or value must be NONE

--dry-run

Report on cleanable locations - not actually cleaning up anything.

-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 check-dates

Synopsis

```
datalad check-dates [-h] [-D DATE] [--rev REVISION] [--annex {all|tree|none}] [--no-tags]
    [--older] [--version] [PATH ...]
```

Description

Find repository dates that are more recent than a reference date.

The main purpose of this tool is to find "leaked" real dates in repositories that are configured to use fake dates. It checks dates from three sources: (1) commit timestamps (author and committer dates), (2) timestamps within files of the "git-annex" branch, and (3) the timestamps of annotated tags.

Options

PATH

Root directory in which to search for Git repositories. The current working directory will be used by default. Constraints: value must be a string or value must be NONE

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

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

-D DATE, --reference-date DATE

Compare dates to this date. If dateutil is installed, this value can be any format that its parser recognizes. Otherwise, it should be a unix timestamp that starts with a "@". The default value corresponds to 01 Jan, 2018 00:00:00 -0000. Constraints: value must be a string [Default: '@1514764800']

--rev REVISION

Search timestamps from commits that are reachable from REVISION. Any revision specification supported by git log, including flags like –all and –tags, can be used. This option can be given multiple times.

--annex {all|tree|none}

Mode for "git-annex" branch search. If 'all', all blobs within the branch are searched. 'tree' limits the search to blobs that are referenced by the tree at the tip of the branch. 'none' disables search of "git-annex" blobs. Constraints: value must be one of ('all', 'tree', 'none') [Default: 'all']

--no-tags

Don't check the dates of annotated tags.

--older

Find dates which are older than the reference date rather than newer.

--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 configuration

Synopsis

```
datalad configuration [-h] [--scope {global|local|branch}] [-d DATASET] [-r] [-R LEVELS]
    [--version] [{dump|get|set|unset}] [name[=value] ...]
```

Description

Get and set dataset, dataset-clone-local, or global configuration

This command works similar to git-config, but some features are not supported (e.g., modifying system configuration), while other features are not available in git-config (e.g., multi-configuration queries).

Query and modification of three distinct configuration scopes is supported:

- 'branch': the persistent configuration in .datalad/config of a dataset branch
- 'local': a dataset clone's Git repository configuration in .git/config
- 'global': non-dataset-specific configuration (usually in $USER/.gitconfig)

Modifications of the persistent 'branch' configuration will not be saved by this command, but have to be committed with a subsequent SAVE call.

Rules of precedence regarding different configuration scopes are the same as in Git, with two exceptions: 1) environment variables can be used to override any datalad configuration, and have precedence over any other configuration scope (see below). 2) the 'branch' scope is considered in addition to the standard git configuration scopes. Its content has lower precedence than Git configuration scopes, but it is committed to a branch, hence can be used to ship (default and branch-specific) configuration with a dataset.

Besides storing configuration settings statically via this command or `git config`, DataLad also reads any DATALAD_* environment on process startup or import, and maps it to a configuration item. Their values take precedence over any other specification. In variable names _ encodes a `.` in the configuration name, and __ encodes a `-`, such that `DATALAD_SOME__VAR` is mapped to `datalad.some-var`. Additionally, a DATALAD_CONFIG_OVERRIDES_JSON

environment variable is queried, which may contain configuration key-value mappings as a JSON-formatted string of a JSON-object:

```
DATALAD_CONFIG_OVERRIDES_JSON='{"datalad.credential.example_com.user": "jane", ...}'
```

This is useful when characters are part of the configuration key that cannot be encoded into an environment variable name. If both individual configuration variables *and* JSON-overrides are used, the former take precedent over the latter, overriding the respective *individual* settings from configurations declared in the JSON-overrides.

This command supports recursive operation for querying and modifying configuration across a hierarchy of datasets.

Examples

Dump the effective configuration, including an annotation for common items:

```
% datalad configuration
```

Query two configuration items:

```
% datalad configuration get user.name user.email
```

Recursively set configuration in all (sub)dataset repositories:

```
% datalad configuration -r set my.config=value
```

Modify the persistent branch configuration (changes are not committed):

```
% datalad configuration --scope branch set my.config=value
```

Options

{dump|get|set|unset}

which action to perform. Constraints: value must be one of ('dump', 'get', 'set', 'unset') [Default: 'dump']

name[=value]

configuration name (for actions 'get' and 'unset'), or name/value pair (for action 'set').

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

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

--scope {global|local|branch}

scope for getting or setting configuration. If no scope is declared for a query, all configuration sources (including overrides via environment variables) are considered according to the normal rules of precedence. For action 'get' only 'branch' and 'local' (which include 'global' here) are supported. For action 'dump', a scope selection is ignored and all available scopes are considered. Constraints: value must be one of ('global', 'local', 'branch')

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

specify the dataset to query or to configure. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-r, --recursive

if set, recurse into potential subdatasets.

-R LEVELS, --recursion-limit LEVELS

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

--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 create-test-dataset

Synopsis

```
datalad create-test-dataset [-h] [--spec SPEC] [--seed SEED] [--version] path
```

Description

Create test (meta-)dataset.

Options

path

path/name where to create (if specified, must not exist). Constraints: value must be a string or value must be NONE

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

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

--spec *SPEC*

spec for hierarchy, defined as a min-max (min could be omitted to assume 0) defining how many (random number from min to max) of sub-datasets to generate at any given level of the hierarchy. Each level separated from each other with /. Example: 1-3/-2 would generate from 1 to 3 subdatasets at the top level, and up to two within those at the 2nd level. Constraints: value must be a string or value must be NONE

--seed *SEED*

seed for rng. 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 download-url

Synopsis

```
datalad download-url [-h] [-d PATH] [-O PATH] [-o] [--archive] [--nosave] [-m MESSAGE]
    [--version] url [url ...]
```

Description

Download content

It allows for a uniform download interface to various supported URL schemes (see command help for details), re-using or asking for authentication details maintained by datalad.

Examples

Download files from an http and S3 URL:

```
% datalad download-url http://example.com/file.dat s3://bucket/file2.dat
```

Download a file to a path and provide a commit message:

```
% datalad download-url -m 'added a file' -O myfile.dat \
  s3://bucket/file2.dat
```

Append a trailing slash to the target path to download into a specified directory:

```
% datalad download-url --path=data/ http://example.com/file.dat
```

Leave off the trailing slash to download into a regular file:

```
% datalad download-url --path=data http://example.com/file.dat
```

Options

url

URL(s) to be downloaded. Supported protocols: 'ftp', 'http', 'https', 's3', 'shub'. Constraints: value must be a string

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

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

-d PATH, --dataset PATH

specify the dataset to add files to. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Use –nosave to prevent adding files to the dataset. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

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

target for download. If the path has a trailing separator, it is treated as a directory, and each specified URL is downloaded under that directory to a base name taken from the URL. Without a trailing separator, the value specifies the name of the downloaded file (file name extensions inferred from the URL may be added to it, if they are not yet present) and only a single URL should be given. In both cases, leading directories will be created if needed. This argument defaults to the current directory. Constraints: value must be a string or value must be NONE

-o, --overwrite

flag to overwrite it if target file exists.

--archive

pass the downloaded files to datalad add-archive-content –delete.

--nosave

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

-m MESSAGE, --message MESSAGE

a description of the state or the changes made to a dataset. Constraints: value must be a string 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 foreach-dataset

Synopsis

```
datalad foreach-dataset [-h] [--cmd-type {auto|external|exec|eval}] [-d DATASET] [--state
    {present|absent|any}] [-r] [-R LEVELS] [--contains PATH]
    [--bottomup] [-s] [--output-streams
    {capture|pass-through|relpath}] [--chpwd {ds|pwd}]
    [--safe-to-consume {auto|all-subds-done|superds-done|always}]
    [-J NJOBS] [--version] ...
```

Description

Run a command or Python code on the dataset and/or each of its sub-datasets.

This command provides a convenience for the cases were no dedicated DataLad command is provided to operate across the hierarchy of datasets. It is very similar to *git submodule foreach* command with the following major differences

- by default (unless –subdatasets-only) it would include operation on the original dataset as well,
- subdatasets could be traversed in bottom-up order,
- can execute commands in parallel (see JOBS option), but would account for the order, e.g. in bottom-up order command is executed in super-dataset only after it is executed in all subdatasets.

Additional notes:

- for execution of "external" commands we use the environment used to execute external git and git-annex commands.

Command format

–cmd-type external: A few placeholders are supported in the command via Python format specification:

- "{pwd}" will be replaced with the full path of the current working directory.
- "{ds}" and "{refds}" will provide instances of the dataset currently operated on and the reference "context" dataset which was provided via `dataset` argument.
- "{tmpdir}" will be replaced with the full path of a temporary directory.

Examples

Aggressively git clean all datasets, running 5 parallel jobs:

```
% datalad foreach-dataset -r -J 5 git clean -dfx
```

Options

COMMAND

command for execution. A leading '–' can be used to disambiguate this command from the preceding options to DataLad. For –cmd-type exec or eval only a single command argument (Python code) is supported.

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

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

--cmd-type {auto|external|exec|eval}

type of the command. EXTERNAL: to be run in a child process using dataset's runner; 'exec': Python source code to execute using 'exec(), no value returned; 'eval': Python source code to evaluate using 'eval()', return value is placed into 'result' field. 'auto': If used via Python API, and *cmd* is a Python function, it will use 'eval', and otherwise would assume 'external'. Constraints: value must be one of ('auto', 'external', 'exec', 'eval') [Default: 'auto']

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

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

--state {present|absent|any}

indicate which (sub)datasets to consider: either only locally present, absent, or any of those two kinds. Constraints: value must be one of ('present', 'absent', 'any') [Default: 'present']

-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

--contains PATH

limit to the subdatasets containing the given path. If a root path of a subdataset is given, the last considered dataset will be the subdataset itself. This option can be given multiple times, in which case datasets that contain any of the given paths will be considered. Constraints: value must be a string or value must be NONE

--bottomup

whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down.

-s, --subdatasets-only

whether to exclude top level dataset. It is implied if a non-empty CONTAINS is used.

--output-streams {capture|pass-through|relpath}, --o-s {capture|pass-through|relpath}

ways to handle outputs. 'capture' and return outputs from 'cmd' in the record ('stdout', 'stderr'); 'pass-through' to the screen (and thus absent from returned record); prefix with 'relpath' captured output (similar to like grep does) and write to stdout and stderr. In 'relpath', relative path is relative to the top of the dataset if DATASET is specified, and if not - relative to current directory. Constraints: value must be one of ('capture', 'pass-through', 'relpath') [Default: 'pass-through']

--chpwd {ds|pwd}

'ds' will change working directory to the top of the corresponding dataset. With 'pwd' no change of working directory will happen. Note that for Python commands, due to use of threads, we do not allow chdir=ds to be used with jobs > 1. Hint: use 'ds' and 'refds' objects' methods to execute commands in the context of those datasets. Constraints: value must be one of ('ds', 'pwd') [Default: 'ds']

--safe-to-consume {auto|all-subds-done|superds-done|always}

Important only in the case of parallel (jobs greater than 1) execution. 'all- subds-done' instructs to not consider super-dataset until command finished execution in all subdatasets (it is the value in case of 'auto' if traversal is bottomup). 'superds-done' instructs to not process subdatasets until command finished in the super-dataset (it is the value in case of 'auto' in traversal is not bottom up, which is the default). With 'always' there is no constraint on either to execute in sub or super dataset. Constraints: value must be one of ('auto', 'all-subds-done', 'superds-done', 'always') [Default: 'auto']

-J NJOBS, --jobs NJOBS

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

--version

show the module and its version which provides the command

Authors

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

datalad sshrun

Synopsis

```
datalad sshrun [-h] [-p PORT] [-4] [-6] [-o OPTION] [-n] [--version] login cmd
```

Description

Run command on remote machines via SSH.

This is a replacement for a small part of the functionality of SSH. In addition to SSH alone, this command can make use of datalad's SSH connection management. Its primary use case is to be used with Git as 'core.sshCommand' or via "GIT_SSH_COMMAND".

Configure *datalad.ssh.identityfile* to pass a file to the ssh's -i option.

Options

login

[user@]hostname.

cmd

command for remote execution.

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

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

-p *PORT*, --port *PORT*

port to connect to on the remote host.

-4

use IPv4 addresses only.

-6

use IPv6 addresses only.

-o OPTION

configuration option passed to SSH.

-n

Do not connect stdin to the process.

--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 shell-completion

Synopsis

```
datalad shell-completion [-h] [--version]
```

Description

Display shell script for enabling shell completion for DataLad.

Output of this command should be "sourced" by the bash or zsh to enable shell completions provided by argcomplete.

Example:

$ source <(datalad shell-completion) $ datalad –<PRESS TAB to display available option>

Options

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

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

--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 wtf

Synopsis

```
datalad wtf [-h] [-d DATASET] [-s {some|all}] [-S SECTION] [--flavor {full|short}]
    [-D {html_details}] [-c] [--version]
```

Description

Generate a report about the DataLad installation and configuration

IMPORTANT: Sharing this report with untrusted parties (e.g. on the web) should be done with care, as it may include identifying information, and/or credentials or access tokens.

Options

-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 report on. no dataset is given, an attempt is made to identify the dataset based on the current working directory. Constraints: Value must be a Dataset or a valid identifier of a Dataset (e.g. a path) or value must be NONE

-s {some|all}, --sensitive {some|all}

if set to 'some' or 'all', it will display sections such as config and metadata which could potentially contain sensitive information (credentials, names, etc.). If 'some', the fields which are known to be sensitive will still be masked out. Constraints: value must be one of ('some', 'all')

-S SECTION, --section SECTION

section to include. If not set - depends on flavor. '*' could be used to force all sections. If there are subsections like section.subsection available, then specifying just 'section' would select all subsections for that section. This option can be given multiple times. Constraints: value must be one of ('configuration', 'credentials', 'datalad', 'dataset', 'dependencies', 'environment', 'extensions', 'git-annex', 'location', 'metadata', 'metadata.extractors', 'metadata.filters', 'metadata.indexers', 'python', 'system', '*')

--flavor {full|short}

Flavor of WTF. 'full' would produce markdown with exhaustive list of sections. 'short' will provide a condensed summary only of datalad and dependencies by default. Use –section to list other sections. Constraints: value must be one of ('full', 'short') [Default: 'full']

-D {html_details}, --decor {html_details}

decoration around the rendering to facilitate embedding into issues etc, e.g. use 'html_details' for posting collapsible entry to GitHub issues. Constraints: value must be one of ('html_details',)

-c, --clipboard

if set, do not print but copy to clipboard (requires pyperclip module).

--version

show the module and its version which provides the command

Authors

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

Deprecated commands

datalad uninstall

Synopsis

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

Description

DEPRECATED: use the DROP command

Options

PATH

path/name of the component to be uninstalled. Constraints: value must be a string or value must be NONE

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

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

-d DATASET, --dataset DATASET

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

-r, --recursive

if set, recurse into potential subdatasets.

--nocheck

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

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

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

--version

show the module and its version which provides the command

Authors

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

1.5.2 Python module reference

This module reference extends the manual with a comprehensive overview of the available functionality built into datalad. Each module in the package is documented by a general summary of its purpose and the list of classes and functions it provides.

High-level user interface

Dataset operations

`api.Dataset`(*args, **kwargs)	Representation of a DataLad dataset/repository
`api.create`([path, initopts, force, ...])	Create a new dataset from scratch.
`api.create_sibling`(sshurl, *[, name, ...])	Create a dataset sibling on a UNIX-like Shell (local or SSH)-accessible machine
`api.create_sibling_github`(reponame, *[, ...])	Create dataset sibling on GitHub.org (or an enterprise deployment).
`api.create_sibling_gitlab`([path, site, ...])	Create dataset sibling at a GitLab site
`api.create_sibling_gogs`(reponame, *[, api, ...])	Create a dataset sibling on a GOGS site
`api.create_sibling_gitea`(reponame, *[, ...])	Create a dataset sibling on a Gitea site
`api.create_sibling_gin`(reponame, *[, ...])	Create a dataset sibling on a GIN site (with content hosting)
`api.create_sibling_ria`(url, name, *[, ...])	Creates a sibling to a dataset in a RIA store
`api.drop`([path, what, reckless, dataset, ...])	Drop content of individual files or entire (sub)datasets
`api.get`([path, source, dataset, recursive, ...])	Get any dataset content (files/directories/subdatasets).
`api.install`([path, source, dataset, ...])	Install one or many datasets from remote URL(s) or local PATH source(s).
`api.push`([path, dataset, to, since, data, ...])	Push a dataset to a known sibling.
`api.remove`([path, dataset, drop, reckless, ...])	Remove components from datasets
`api.save`([path, message, dataset, ...])	Save the current state of a dataset
`api.status`([path, dataset, annex, ...])	Report on the state of dataset content.
`api.update`([path, sibling, merge, how, ...])	Update a dataset from a sibling.
`api.unlock`([path, dataset, recursive, ...])	Unlock file(s) of a dataset

datalad.api.Dataset

class datalad.api.**Dataset**(**args*, ***kwargs*)

Representation of a DataLad dataset/repository

This is the core data type of DataLad: a representation of a dataset. At its core, datasets are (git-annex enabled) Git repositories. This class provides all operations that can be performed on a dataset.

Creating a dataset instance is cheap, all actual operations are delayed until they are actually needed. Creating multiple *Dataset* class instances for the same Dataset location will automatically yield references to the same object.

A dataset instance comprises of two major components: a *repo* attribute, and a *config* attribute. The former offers access to low-level functionality of the Git or git-annex repository. The latter gives access to a dataset's configuration manager.

Most functionality is available via methods of this class, but also as stand-alone functions with the same name in *datalad.api*.

__init__(*path*)

Parameters

path (*str or Path*) – Path to the dataset location. This location may or may not exist yet.

Methods

__init__(path) **type path**	
add_archive_content(*[, dataset, annex, ...])	Add content of an archive under git annex control.
add_readme(*[, dataset, existing])	Add basic information about DataLad datasets to a README file
addurls(urlformat, filenameformat, *[, ...])	Create and update a dataset from a list of URLs.
clean(*[, what, dry_run, recursive, ...])	Clean up after DataLad (possible temporary files etc.)
clone([path, git_clone_opts, dataset, ...])	Obtain a dataset (copy) from a URL or local directory
close()	Perform operations which would close any possible process using this Dataset
configuration([spec, scope, dataset, ...])	Get and set dataset, dataset-clone-local, or global configuration
copy_file(*[, dataset, recursive, ...])	Copy files and their availability metadata from one dataset to another.
create([initopts, force, description, ...])	Create a new dataset from scratch.
create_sibling(*[, name, target_dir, ...])	Create a dataset sibling on a UNIX-like Shell (local or SSH)-accessible machine
create_sibling_gin(*[, dataset, recursive, ...])	Create a dataset sibling on a GIN site (with content hosting)
create_sibling_gitea(*[, dataset, ...])	Create a dataset sibling on a Gitea site
create_sibling_github(*[, dataset, ...])	Create dataset sibling on GitHub.org (or an enterprise deployment).
create_sibling_gitlab(*[, site, project, ...])	Create dataset sibling at a GitLab site
create_sibling_gogs(*[, api, dataset, ...])	Create a dataset sibling on a GOGS site
create_sibling_ria(name, *[, dataset, ...])	Creates a sibling to a dataset in a RIA store
diff(*[, fr, to, dataset, annex, untracked, ...])	Report differences between two states of a dataset (hierarchy)
download_url(*[, dataset, path, overwrite, ...])	Download content
drop(*[, what, reckless, dataset, ...])	Drop content of individual files or entire (sub)datasets
export_archive(*[, dataset, archivetype, ...])	Export the content of a dataset as a TAR/ZIP archive.
export_archive_ora([opts, dataset, remote, ...])	Export an archive of a local annex object store for the ORA remote.
export_to_figshare(*[, dataset, ...])	Export the content of a dataset as a ZIP archive to figshare
foreach_dataset(*[, cmd_type, dataset, ...])	Run a command or Python code on the dataset and/or each of its sub-datasets.
get(*[, source, dataset, recursive, ...])	Get any dataset content (files/directories/subdatasets).
get_superdataset([datalad_only, topmost, ...])	Get the dataset's superdataset
install(*[, source, dataset, get_data, ...])	Install one or many datasets from remote URL(s) or local PATH source(s).
is_installed()	Returns whether a dataset is installed.
no_annex(pattern[, ref_dir, makedirs])	Configure a dataset to never put some content into the dataset's annex
push(*[, dataset, to, since, data, force, ...])	Push a dataset to a known sibling.
recall_state(whereto)	Something that can be used to checkout a particular state (tag, commit) to "undo" a change or switch to a otherwise desired previous state.

continues on next page

Table 1 – continued from previous page

`remove`(*[, dataset, drop, reckless, ...])	Remove components from datasets
`rerun`(*[, since, dataset, branch, message, ...])	Re-execute previous *datalad run* commands.
`run`(*[, dataset, inputs, outputs, expand, ...])	Run an arbitrary shell command and record its impact on a dataset.
`run_procedure`(*[, dataset, discover, help_proc])	Run prepared procedures (DataLad scripts) on a dataset
`save`(*[, message, dataset, version_tag, ...])	Save the current state of a dataset
`siblings`(*[, dataset, name, url, pushurl, ...])	Manage sibling configuration
`status`(*[, dataset, annex, untracked, ...])	Report on the state of dataset content.
`subdatasets`(*[, dataset, state, fulfilled, ...])	Report subdatasets and their properties.
`uninstall`(*[, dataset, recursive, check, ...])	DEPRECATED: use the *drop* command
`unlock`(*[, dataset, recursive, recursion_limit])	Unlock file(s) of a dataset
`update`(*[, sibling, merge, how, how_subds, ...])	Update a dataset from a sibling.
`wtf`(*[, sensitive, sections, flavor, decor, ...])	Generate a report about the DataLad installation and configuration

Attributes

`config`	Get a `ConfigManager` instance for a dataset's configuration
`id`	Identifier of the dataset.
`path`	path to the dataset
`pathobj`	pathobj for the dataset
`repo`	Get an instance of the version control system/repo for this dataset, or None if there is none yet (or none anymore).

datalad.api.create

`datalad.api.`**`create`**(*path=None*, *initopts=None*, *, *force=False*, *description=None*, *dataset=None*, *annex=True*, *fake_dates=False*, *cfg_proc=None*)

Create a new dataset from scratch.

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

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

Plain Git repositories can be created via *annex=False*. However, the result will not be a full dataset, and, consequently, not all features are supported (e.g. a description).

To create a local version of a remote dataset use the ~datalad.api.install command instead.

Note: Power-user info: This command uses git init and git annex init to prepare the new dataset. Registering to

a superdataset is performed via a git submodule add operation in the discovered superdataset.

Examples

Create a dataset 'mydataset' in the current directory:

```
> create(path='mydataset')
```

Apply the text2git procedure upon creation of a dataset:

```
> create(path='mydataset', cfg_proc='text2git')
```

Create a subdataset in the root of an existing dataset:

```
> create(dataset='.', path='mysubdataset')
```

Create a dataset in an existing, non-empty directory:

```
> create(force=True)
```

Create a plain Git repository:

```
> create(path='mydataset', annex=False)
```

Parameters

- **path** (*str or Dataset or None, optional*) – path where the dataset shall be created, directories will be created as necessary. If no location is provided, a dataset will be created in the location specified by *dataset* (if given) or the current working directory. Either way the command will error if the target directory is not empty. Use *force* to create a dataset in a non- empty directory. [Default: None]
- **initopts** – options to pass to git init. Options can be given as a list of command line arguments or as a GitPython-style option dictionary. Note that not all options will lead to viable results. For example ' –bare' will not yield a repository where DataLad can adjust files in its working tree. [Default: None]
- **force** (*bool, optional*) – enforce creation of a dataset in a non-empty directory. [Default: False]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the create operation on. If a dataset is given along with *path*, a new subdataset will be created in it at the *path* provided to the create command. If a dataset is given but *path* is unspecified, a new dataset will be created at the location specified by this option. [Default: None]
- **annex** (*bool, optional*) – if disabled, a plain Git repository will be created without any annex. [Default: True]
- **fake_dates** (*bool, optional*) – Configure the repository to use fake dates. The date for a new commit will be set to one second later than the latest commit in the repository. This can be used to anonymize dates. [Default: False]

- **cfg_proc** – Run cfg_PROC procedure(s) (can be specified multiple times) on the created dataset. Use *run_procedure(discover=True)* to get a list of available procedures, such as cfg_text2git. [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: constraint:(action:{create} or status:{ok, notneeded})]
- **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: 'datasets']
- **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: 'item-or-list']

datalad.api.create_sibling

datalad.api.**create_sibling**(*sshurl*, **, *name=None*, *target_dir=None*, *target_url=None*, *target_pushurl=None*, *dataset=None*, *recursive=False*, *recursion_limit=None*, *existing='error'*, *shared=None*, *group=None*, *ui=False*, *as_common_datasrc=None*, *publish_by_default=None*, *publish_depends=None*, *annex_wanted=None*, *annex_group=None*, *annex_groupwanted=None*, *inherit=False*, *since=None*)

Create a dataset sibling on a UNIX-like Shell (local or SSH)-accessible machine

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

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

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

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

Parameters

- **sshurl** (*str*) – Login information for the target server. This can be given as a URL (ssh://host/path), SSH-style (user@host:path) or just a local path. Unless overridden, this also serves the future dataset's access URL and path on the server.
- **name** (*str or None, optional*) – sibling name to create for this publication target. If *recursive* is set, the same name will be used to label all the subdatasets' siblings. When creating a target dataset fails, no sibling is added. [Default: None]
- **target_dir** (*str or None, optional*) – path to the directory *on the server* where the dataset shall be created. By default this is set to the URL (or local path) specified via *sshurl*. If a relative path is provided here, it is interpreted as being relative to the user's home directory on the server (or relative to *sshurl*, when that is a local path). Additional features are relevant for recursive processing of datasets with subdatasets. By default, the local dataset structure is replicated on the server. However, it is possible to provide a template for generating different target directory names for all (sub)datasets. Templates can contain certain placeholder that are substituted for each (sub)dataset. For example: "/mydirectory/dataset%%RELNAME". Supported placeholders: %%RELNAME - the name of the datasets, with any slashes replaced by dashes. [Default: None]
- **target_url** (*str or None, optional*) – "public" access URL of the to-be-created target dataset(s) (default: *sshurl*). Accessibility of this URL determines the access permissions of potential consumers of the dataset. As with *target_dir*, templates (same set of placeholders) are supported. Also, if specified, it is provided as the annex description. [Default: None]
- **target_pushurl** (*str or None, optional*) – In case the *target_url* cannot be used to publish to the dataset, this option specifies an alternative URL for this purpose. As with *target_url*, templates (same set of placeholders) are supported. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to create the publication target for. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **existing** (*{'skip', 'error', 'reconfigure', 'replace'}, optional*) – action to perform, if a sibling is already configured under the given name and/or a target (non-empty) directory already exists. In this case, a dataset can be skipped ('skip'), the sibling configuration be updated ('reconfigure'), or process interrupts with error ('error'). DANGER ZONE: If 'replace' is used, an existing target directory will be forcefully removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). *replace* could lead to data loss, so use with care. To minimize possibility of data loss, in interactive mode DataLad will ask for confirmation, but it would raise an exception in non-interactive mode. [Default: 'error']
- **shared** (*str or bool or None, optional*) – if given, configures the access permissions on the server for multi- users (this could include access by a webserver!). Possible

values for this option are identical to those of *git init –shared* and are described in its documentation. [Default: None]

- **group** (*str or None, optional*) – Filesystem group for the repository. Specifying the group is particularly important when shared="group". [Default: None]
- **ui** (*bool or str, optional*) – publish a web interface for the dataset with an optional user- specified name for the html at publication target. defaults to *index.html* at dataset root. [Default: False]
- **as_common_datasrc** – configure the created sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset (technical: git-annex auto-enabled special remote). [Default: None]
- **publish_by_default** (*list of str or None, optional*) – add a refspec to be published to this sibling by default if nothing specified. [Default: None]
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **annex_wanted** (*str or None, optional*) – expression to specify 'wanted' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-wanted/ for more information. [Default: None]
- **annex_group** (*str or None, optional*) – expression to specify a group for the repository. See https://git- annex.branchable.com/git-annex-group/ for more information. [Default: None]
- **annex_groupwanted** (*str or None, optional*) – expression for the groupwanted. Makes sense only if annex_wanted="groupwanted" and annex-group is given too. See https://git-annex.branchable.com/git-annex-groupwanted/ for more information. [Default: None]
- **inherit** (*bool, optional*) – if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: False]
- **since** (*str or None, optional*) – limit processing to subdatasets that have been changed since a given state (by tag, branch, commit, etc). This can be used to create siblings for recently added subdatasets. If '^' is given, the last state of the current branch at the sibling is taken as a starting point. [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.api.create_sibling_github

datalad.api.**create_sibling_github**(*reponame*, *, *dataset=None*, *recursive=False*, *recursion_limit=None*, *name='github'*, *existing='error'*, *github_login=None*, *credential=None*, *github_organization=None*, *access_protocol='https'*, *publish_depends=None*, *private=False*, *description=None*, *dryrun=False*, *dry_run=False*, *api='https://api.github.com'*)

Create dataset sibling on GitHub.org (or an enterprise deployment).

GitHub is a popular commercial solution for code hosting and collaborative development. GitHub cannot host dataset content (but see LFS, http://handbook.datalad.org/r.html?LFS). However, in combination with other data sources and siblings, publishing a dataset to GitHub can facilitate distribution and exchange, while still allowing any dataset consumer to obtain actual data content from alternative sources.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Settings->Developer Settings->Personal access tokens->Generate new token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Github instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

Changed in version 0.16: The API has been aligned with the some `create_sibling_...` commands of other GitHub-like services, such as GOGS, GIN, GitTea.

Deprecated since version 0.16: The `dryrun` option will be removed in a future release, use the renamed `dry_run` option instead. The `github_login` option will be removed in a future release, use the `credential` option instead. The `github_organization` option will be removed in a future release, prefix the reposity name with `<org>/` instead.

Examples

Use a new sibling on GIN as a common data source that is auto- available when cloning from GitHub:

```
> ds = Dataset('.')

# the sibling on GIN will host data content
> ds.create_sibling_gin('myrepo', name='gin')

# the sibling on GitHub will be used for collaborative work
> ds.create_sibling_github('myrepo', name='github')

# register the storage of the public GIN repo as a data source
> ds.siblings('configure', name='gin', as_common_datasrc='gin-storage')

# announce its availability on github
> ds.push(to='github')
```

Parameters

- **reponame** (*str*) – repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset.
- **dataset** (*Dataset or None, optional*) – dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **name** (*str or None, optional*) – name of the sibling in the local dataset installation (remote name). [Default: 'github']
- **existing** (*{'skip', 'error', 'reconfigure', 'replace'}, optional*) – behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). *replace* could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. [Default: 'error']
- **github_login** (*str or None, optional*) – Deprecated, use the credential parameter instead. If given must be a personal access token. [Default: None]
- **credential** (*str or None, optional*) – name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). [Default: None]
- **github_organization** (*str or None, optional*) – Deprecated, prepend a repo name with an '<orgname>/' prefix instead. [Default: None]

- **access_protocol** (*{'https', 'ssh', 'https-ssh'}, optional*) – access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. [Default: 'https']
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **private** (*bool, optional*) – if set, create a private repository. [Default: False]
- **description** (*str or None, optional*) – Brief description, displayed on the project's page. [Default: None]
- **dryrun** (*bool, optional*) – Deprecated. Use the renamed `dry_run` parameter. [Default: False]
- **dry_run** (*bool, optional*) – if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets. [Default: False]
- **api** (*str or None, optional*) – URL of the GitHub instance API. [Default: 'https://api.github.com']
- **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.api.create_sibling_gitlab

datalad.api.**create_sibling_gitlab**(*path=None*, *, *site=None*, *project=None*, *layout=None*, *dataset=None*, *recursive=False*, *recursion_limit=None*, *name=None*, *existing='error'*, *access=None*, *publish_depends=None*, *description=None*, *dryrun=False*, *dry_run=False*)

Create dataset sibling at a GitLab site

An existing GitLab project, or a project created via the GitLab web interface can be configured as a sibling with the siblings command. Alternatively, this command can create a GitLab project at any location/path a given user has appropriate permissions for. This is particularly helpful for recursive sibling creation for subdatasets. API access and authentication are implemented via python-gitlab, and all its features are supported. A particular GitLab site must be configured in a named section of a python-gitlab.cfg file (see https://python-gitlab.readthedocs.io/en/stable/cli-usage.html#configuration-file-format for details), such as:

```
[mygit]
url = https://git.example.com
api_version = 4
private_token = abcdefghijklmnopqrst
```

Subsequently, this site is identified by its name ('mygit' in the example above).

(Recursive) sibling creation for all, or a selected subset of subdatasets is supported with two different project layouts (see –layout):

"flat"
All datasets are placed as GitLab projects in the same group. The project name of the top-level dataset follows the configured datalad.gitlab-SITENAME-project configuration. The project names of contained subdatasets extend the configured name with the subdatasets' s relative path within the root dataset, with all path separator characters replaced by '-'. This path separator is configurable (see Configuration).

"collection"
A new group is created for the dataset hierarchy, following the datalad.gitlab-SITENAME-project configuration. The root dataset is placed in a "project" project inside this group, and all nested subdatasets are represented inside the group using a "flat" layout. The root datasets project name is configurable (see Configuration).

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

Configuration

Many configuration switches and options for GitLab sibling creation can be provided as arguments to the command. However, it is also possible to specify a particular setup in a dataset's configuration. This is particularly important when managing large collections of datasets. Configuration options are:

"datalad.gitlab-default-site"
Name of the default GitLab site (see –site)

"datalad.gitlab-SITENAME-siblingname"
Name of the sibling configured for the local dataset that points to the GitLab instance SITENAME (see

–name)

"datalad.gitlab-SITENAME-layout"
Project layout used at the GitLab instance SITENAME (see –layout)

"datalad.gitlab-SITENAME-access"
Access method used for the GitLab instance SITENAME (see –access)

"datalad.gitlab-SITENAME-project"
Project "location/path" used for a datasets at GitLab instance SITENAME (see –project). Configuring this is useful for deriving project paths for subdatasets, relative to superdataset. The root-level group ("location") needs to be created beforehand via GitLab's web interface.

"datalad.gitlab-default-projectname"
The collection layout publishes (sub)datasets as projects with a custom name. The default name "project" can be overridden with this configuration.

"datalad.gitlab-default-pathseparator"
The flat and collection layout represent subdatasets with project names that correspond to their path within the superdataset, with the regular path separator replaced with a "-": superdataset-subdataset. This configuration can be used to override this default separator.

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gitlab instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

Parameters

- **path** – selectively create siblings for any datasets underneath a given path. By default only the root dataset is considered. [Default: None]
- **site** (*None or str, optional*) – name of the GitLab site to create a sibling at. Must match an existing python-gitlab configuration section with location and authentication settings (see https://python- gitlab.readthedocs.io/en/stable/cli-usage.html#configuration). By default the dataset configuration is consulted. [Default: None]
- **project** (*None or str, optional*) – project name/location at the GitLab site. If a subdataset of the reference dataset is processed, its project path is automatically determined by the *layout* configuration, by default. Users need to create the root-level GitLab group (NAME) via the webinterface before running the command. [Default: None]
- **layout** (*{None, 'collection', 'flat'}, optional*) – layout of projects at the GitLab site, if a collection, or a hierarchy of datasets and subdatasets is to be created. By default the dataset configuration is consulted. [Default: None]
- **dataset** (*Dataset or None, optional*) – reference or root dataset. If no path constraints are given, a sibling for this dataset will be created. In this and all other cases, the reference dataset is also consulted for the GitLab configuration, and desired project layout. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]

- **name** (*str or None, optional*) – name to represent the GitLab sibling remote in the local dataset installation. If not specified a name is looked up in the dataset configuration, or defaults to the *site* name. [Default: None]
- **existing** (*{'skip', 'error', 'reconfigure'}, optional*) – desired behavior when already existing or configured siblings are discovered. 'skip': ignore; 'error': fail, if access URLs differ; 'reconfigure': use the existing repository and reconfigure the local dataset to use it as a sibling. [Default: 'error']
- **access** (*{None, 'http', 'ssh', 'ssh+http'}, optional*) – access method used for data transfer to and from the sibling. 'ssh': read and write access used the SSH protocol; 'http': read and write access use HTTP requests; 'ssh+http': read access is done via HTTP and write access performed with SSH. Dataset configuration is consulted for a default, 'http' is used otherwise. [Default: None]
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **description** (*str or None, optional*) – brief description for the GitLab project (displayed on the site). [Default: None]
- **dryrun** (*bool, optional*) – Deprecated. Use the renamed `dry_run` parameter. [Default: False]
- **dry_run** (*bool, optional*) – if set, no repository will be created, only tests for name collisions will be performed, and would-be repository names are reported for all relevant datasets. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead.

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

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

datalad.api.create_sibling_gogs

datalad.api.**create_sibling_gogs**(*reponame*, ***, *api=None*, *dataset=None*, *recursive=False*, *recursion_limit=None*, *name=None*, *existing='error'*, *credential=None*, *access_protocol='https'*, *publish_depends=None*, *private=False*, *description=None*, *dry_run=False*)

Create a dataset sibling on a GOGS site

GOGS is a self-hosted, free and open source code hosting solution with low resource demands that enable running it on inexpensive devices like a Raspberry Pi, or even directly on a NAS device.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Your Settings->Applications->Generate New Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gogs instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16.

Parameters

- **reponame** (*str*) – repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset.
- **api** (*str or None, optional*) – URL of the GOGS instance without a 'api/<version>' suffix. [Default: None]
- **dataset** (*Dataset or None, optional*) – dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **name** (*str or None, optional*) – name of the sibling in the local dataset installation (remote name). [Default: None]
- **existing** (*{'skip', 'error', 'reconfigure', 'replace'}, optional*) – behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the

sibling (re-)configured (thus implies 'reconfigure'). *replace* could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. [Default: 'error']

- **credential** (*str or None, optional*) – name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). [Default: None]
- **access_protocol** (*{'https', 'ssh', 'https-ssh'}, optional*) – access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. [Default: 'https']
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **private** (*bool, optional*) – if set, create a private repository. [Default: False]
- **description** (*str or None, optional*) – Brief description, displayed on the project's page. [Default: None]
- **dry_run** (*bool, optional*) – if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result

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

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

datalad.api.create_sibling_gitea

datalad.api.**create_sibling_gitea**(*reponame*, *, *dataset=None*, *recursive=False*, *recursion_limit=None*, *name='gitea'*, *existing='error'*, *api='https://gitea.com'*, *credential=None*, *access_protocol='https'*, *publish_depends=None*, *private=False*, *description=None*, *dry_run=False*)

Create a dataset sibling on a Gitea site

Gitea is a lightweight, free and open source code hosting solution with low resource demands that enable running it on inexpensive devices like a Raspberry Pi.

This command uses the main Gitea instance at https://gitea.com as the default target, but other deployments can be used via the 'api' parameter.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Settings->Applications->Generate Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gitea instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16.

Parameters

- **reponame** (*str*) – repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset.
- **dataset** (*Dataset or None, optional*) – dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **name** (*str or None, optional*) – name of the sibling in the local dataset installation (remote name). [Default: 'gitea']
- **existing** (*{'skip', 'error', 'reconfigure', 'replace'}, optional*) – behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the

sibling (re-)configured (thus implies 'reconfigure'). *replace* could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. [Default: 'error']

- **api** (*str or None, optional*) – URL of the Gitea instance without a 'api/<version>' suffix. [Default: 'https://gitea.com']
- **credential** (*str or None, optional*) – name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). [Default: None]
- **access_protocol** (*{'https', 'ssh', 'https-ssh'}, optional*) – access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. [Default: 'https']
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **private** (*bool, optional*) – if set, create a private repository. [Default: False]
- **description** (*str or None, optional*) – Brief description, displayed on the project's page. [Default: None]
- **dry_run** (*bool, optional*) – if set, no repository will be created, only tests for sibling name collisions will be performed, and would-be repository names are reported for all relevant datasets. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths',*

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

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

datalad.api.create_sibling_gin

datalad.api.**create_sibling_gin**(*reponame*, *, *dataset=None*, *recursive=False*, *recursion_limit=None*, *name='gin'*, *existing='error'*, *api='https://gin.g-node.org'*, *credential=None*, *access_protocol='https-ssh'*, *publish_depends=None*, *private=False*, *description=None*, *dry_run=False*)

Create a dataset sibling on a GIN site (with content hosting)

GIN (G-Node infrastructure) is a free data management system. It is a GitHub-like, web-based repository store and provides fine-grained access control to shared data. GIN is built on Git and git-annex, and can natively host DataLad datasets, including their data content!

This command uses the main GIN instance at https://gin.g-node.org as the default target, but other deployments can be used via the 'api' parameter.

An SSH key, properly registered at the GIN instance, is required for data upload via DataLad. Data download from public projects is also possible via anonymous HTTP.

In order to be able to use this command, a personal access token has to be generated on the platform (Account->Your Settings->Applications->Generate New Token).

This command can be configured with "datalad.create-sibling-ghlike.extra-remote-settings.NETLOC.KEY=VALUE" in order to add any local KEY = VALUE configuration to the created sibling in the local *.git/config* file. NETLOC is the domain of the Gin instance to apply the configuration for. This leads to a behavior that is equivalent to calling datalad's `siblings('configure', ...)``||``siblings configure` command with the respective KEY-VALUE pair after creating the sibling. The configuration, like any other, could be set at user- or system level, so users do not need to add this configuration to every sibling created with the service at NETLOC themselves.

New in version 0.16.

Examples

Create a repo 'myrepo' on GIN and register it as sibling 'mygin':

```
> create_sibling_gin('myrepo', name='mygin', dataset='.')
```

Create private repos with name(-prefix) 'myrepo' on GIN for a dataset and all its present subdatasets:

```
> create_sibling_gin('myrepo', dataset='.', recursive=True, private=True)
```

Create a sibling repo on GIN, and register it as a common data source in the dataset that is available regardless of whether the dataset was directly cloned from GIN:

```
> ds = Dataset('.')
> ds.create_sibling_gin('myrepo', name='gin')
# first push creates git-annex branch remotely and obtains annex UUID
> ds.push(to='gin')
> ds.siblings('configure', name='gin', as_common_datasrc='gin-storage')
# announce availability (redo for other siblings)
> ds.push(to='gin')
```

Parameters

- **reponame** (*str*) – repository name, optionally including an '<organization>/' prefix if the repository shall not reside under a user's namespace. When operating recursively, a suffix will be appended to this name for each subdataset.
- **dataset** (*Dataset or None, optional*) – dataset to create the publication target for. If not given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **name** (*str or None, optional*) – name of the sibling in the local dataset installation (remote name). [Default: 'gin']
- **existing** (*{'skip', 'error', 'reconfigure', 'replace'}, optional*) – behavior when already existing or configured siblings are discovered: skip the dataset ('skip'), update the configuration ('reconfigure'), or fail ('error'). DEPRECATED DANGER ZONE: With 'replace', an existing repository will be irreversibly removed, re-initialized, and the sibling (re-)configured (thus implies 'reconfigure'). *replace* could lead to data loss! In interactive sessions a confirmation prompt is shown, an exception is raised in non-interactive sessions. The 'replace' mode will be removed in a future release. [Default: 'error']
- **api** (*str or None, optional*) – URL of the GIN instance without an 'api/<version>' suffix. [Default: 'https://gin.g-node.org']
- **credential** (*str or None, optional*) – name of the credential providing a personal access token to be used for authorization. The token can be supplied via configuration setting 'datalad.credential.<name>.token', or environment variable DATALAD_CREDENTIAL_<NAME>_TOKEN, or will be queried from the active credential store using the provided name. If none is provided, the host-part of the API URL is used as a name (e.g. 'https://api.github.com' -> 'api.github.com'). [Default: None]
- **access_protocol** (*{'https', 'ssh', 'https-ssh'}, optional*) – access protocol/URL to configure for the sibling. With 'https-ssh' SSH will be used for write access, whereas HTTPS is used for read access. [Default: 'https-ssh']
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **private** (*bool, optional*) – if set, create a private repository. [Default: False]
- **description** (*str or None, optional*) – Brief description, displayed on the project's page. [Default: None]

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

datalad.api.create_sibling_ria

datalad.api.**create_sibling_ria**(*url*, *name*, **, *dataset=None*, *storage_name=None*, *alias=None*, *post_update_hook=False*, *shared=None*, *group=None*, *storage_sibling=True*, *existing='error'*, *new_store_ok=False*, *trust_level=None*, *recursive=False*, *recursion_limit=None*, *disable_storage__=None*, *push_url=None*)

Creates a sibling to a dataset in a RIA store

Communication with a dataset in a RIA store is implemented via two siblings. A regular Git remote (repository sibling) and a git-annex special remote for data transfer (storage sibling) – with the former having a publication dependency on the latter. By default, the name of the storage sibling is derived from the repository sibling's name by appending "-storage".

The store's base path is expected to not exist, be an empty directory, or a valid RIA store.

Notes

RIA URL format

Interactions with new or existing RIA stores require RIA URLs to identify the store or specific datasets inside of it.

The general structure of a RIA URL pointing to a store takes the form `ria+[scheme]://<storelocation>` (e.g., `ria+ssh://[user@]hostname:/absolute/path/to/ria-store`, or `ria+file:///absolute/path/to/ria-store`)

The general structure of a RIA URL pointing to a dataset in a store (for example for cloning) takes a similar form, but appends either the datasets UUID or a "~" symbol followed by the dataset's alias name: `ria+[scheme]://<storelocation>#<dataset-UUID>` or `ria+[scheme]://<storelocation>#~<aliasname>`. In addition, specific version identifiers can be appended to the URL with an additional "@" symbol: `ria+[scheme]://<storelocation>#<dataset-UUID>@<dataset-version>`, where `dataset-version` refers to a branch or tag.

RIA store layout

A RIA store is a directory tree with a dedicated subdirectory for each dataset in the store. The subdirectory name is constructed from the DataLad dataset ID, e.g. `124/68afe-59ec-11ea-93d7-f0d5bf7b5561`, where the first three characters of the ID are used for an intermediate subdirectory in order to mitigate files system limitations for stores containing a large number of datasets.

By default, a dataset in a RIA store consists of two components: A Git repository (for all dataset contents stored in Git) and a storage sibling (for dataset content stored in git-annex).

It is possible to selectively disable either component using `storage-sibling 'off'` or `storage-sibling 'only'`, respectively. If neither component is disabled, a dataset's subdirectory layout in a RIA store contains a standard bare Git repository and an `annex/` subdirectory inside of it. The latter holds a Git-annex object store and comprises the storage sibling. Disabling the standard git-remote (`storage-sibling='only'`) will result in not having the bare git repository, disabling the storage sibling (`storage-sibling='off'`) will result in not having the `annex/` subdirectory.

Optionally, there can be a further subdirectory `archives` with (compressed) 7z archives of annex objects. The storage remote is able to pull annex objects from these archives, if it cannot find in the regular annex object store. This feature can be useful for storing large collections of rarely changing data on systems that limit the number of files that can be stored.

Each dataset directory also contains a `ria-layout-version` file that identifies the data organization (as, for example, described above).

Lastly, there is a global `ria-layout-version` file at the store's base path that identifies where dataset subdirectories themselves are located. At present, this file must contain a single line stating the version (currently "1"). This line MUST end with a newline character.

It is possible to define an alias for an individual dataset in a store by placing a symlink to the dataset location into an `alias/` directory in the root of the store. This enables dataset access via URLs of format: `ria+<protocol>://<storelocation>#~<aliasname>`.

Compared to standard git-annex object stores, the `annex/` subdirectories used as storage siblings follow a different layout naming scheme ('dirhashmixed' instead of 'dirhashlower'). This is mostly noted as a technical detail, but also serves to remind git-annex powerusers to refrain from running git-annex commands directly in-store as it can cause severe damage due to the layout difference. Interactions should be handled via the ORA special remote instead.

Error logging

To enable error logging at the remote end, append a pipe symbol and an "l" to the version number in ria-layout-version (like so: `1|l\n`).

Error logging will create files in an "error_log" directory whenever the git-annex special remote (storage sibling) raises an exception, storing the Python traceback of it. The logfiles are named according to the scheme `<dataset id>.<annex uuid of the remote>.log` showing "who" ran into this issue with which dataset. Because logging can potentially leak personal data (like local file paths for example), it can be disabled client-side by setting the configuration variable `annex.ora-remote.<storage-sibling-name>.ignore-remote-config`.

Parameters

- **url** (*str or None*) – URL identifying the target RIA store and access protocol. If `push_url` is given in addition, this is used for read access only. Otherwise it will be used for write access too and to create the repository sibling in the RIA store. Note, that HTTP(S) currently is valid for consumption only thus requiring to provide `push_url`.
- **name** (*str or None*) – Name of the sibling. With *recursive*, the same name will be used to label all the subdatasets' siblings.
- **dataset** (*Dataset or None, optional*) – 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]
- **storage_name** (*str or None, optional*) – 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]
- **alias** (*str or None, optional*) – Alias for the dataset in the RIA store. Add the necessary symlink so that this dataset can be cloned from the RIA store using the given ALIAS instead of its ID. With *recursive=True*, only the top dataset will be aliased. [Default: None]
- **post_update_hook** (*bool, optional*) – Enable Git's default post-update-hook for the created sibling. This is useful when the sibling is made accessible via a "dumb server" that requires running 'git update-server-info' to let Git interact properly with it. [Default: False]
- **shared** (*str or bool or None, optional*) – If given, configures the permissions in the RIA store for multi- users access. Possible values for this option are identical to those of *git init –shared* and are described in its documentation. [Default: None]
- **group** (*str or None, optional*) – Filesystem group for the repository. Specifying the group is crucial when shared="group". [Default: None]
- **storage_sibling** (*{'only'} or bool or None, optional*) – By default, an ORA storage sibling and a Git repository sibling are created (True|'on'). Alternatively, creation of the storage sibling can be disabled (False|'off'), or a storage sibling created only and no Git sibling ('only'). In the latter mode, no Git installation is required on the target host. [Default: True]
- **existing** (*{'skip', 'error', 'reconfigure'}, optional*) – Action to perform, if a (storage) sibling is already configured under the given name and/or a target already exists. In this case, a dataset can be skipped ('skip'), an existing target repository be forcefully re-initialized, and the sibling (re-)configured ('reconfigure'), or the command be instructed to fail ('error'). [Default: 'error']
- **new_store_ok** (*bool, optional*) – When set, a new store will be created, if necessary. Otherwise, a sibling will only be created if the url points to an existing RIA store. [Default: False]
- **trust_level** (*{'trust', 'semitrust', 'untrust', None}, optional*) – specify a trust level for the storage sibling. If not specified, the default git-annex trust level is used.

'trust' should be used with care (see the git-annex-trust man page). [Default: None]

- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **disable_storage** (*bool, optional*) – This option is deprecated. Use '–storage-sibling off' instead. [Default: None]
- **push_url** (*str or None, optional*) – URL identifying the target RIA store and access protocol for write access to the storage sibling. If given this will also be used for creation of the repository sibling in the RIA store. [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.api.drop

datalad.api.**drop**(*path=None*, **, what='filecontent', reckless=None, dataset=None, recursive=False, recursion_limit=None, jobs=None, check=None, if_dirty=None*)

Drop content of individual files or entire (sub)datasets

This command is the antagonist of 'get'. It can undo the retrieval of file content, and the installation of subdatasets.

Dropping is a safe-by-default operation. Before dropping any information, the command confirms the continued availability of file-content (see e.g., configuration 'annex.numcopies'), and the state of all dataset branches from at least one known dataset sibling. Moreover, prior removal of an entire dataset annex, that it is confirmed that it is no longer marked as existing in the network of dataset siblings.

Importantly, all checks regarding version history availability and local annex availability are performed using the current state of remote siblings as known to the local dataset. This is done for performance reasons and for resilience in case of absent network connectivity. To ensure decision making based on up-to-date information, it is advised to execute a dataset update before dropping dataset components.

Examples

Drop single file content:

```
> drop('path/to/file')
```

Drop all file content in the current dataset:

```
> drop('.')
```

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

```
> drop(dataset='.', recursive=True)
```

Disable check to ensure the configured minimum number of remote sources for dropped data:

```
> drop(path='path/to/content', reckless='availability')
```

Drop (uninstall) an entire dataset (will fail with subdatasets present):

```
> drop(what='all')
```

Kill a dataset recklessly with any existing subdatasets too(this will be fast, but will disable any and all safety checks):

```
> drop(what='all', reckless='kill', recursive=True)
```

Parameters

- **path** (*sequence of str or None, optional*) – path of a dataset or dataset component to be dropped. [Default: None]
- **what** (*{'filecontent', 'allkeys', 'datasets', 'all'}, optional*) – select what type of items shall be dropped. With 'filecontent', only the file content (git-annex keys) of files in a dataset's worktree will be dropped. With 'allkeys', content of any version of any file in any branch (including, but not limited to the worktree) will be dropped. This effectively empties the annex of a local dataset. With 'datasets', only complete datasets will be dropped (implies 'allkeys' mode for each such dataset), but no filecontent will be dropped for any files

in datasets that are not dropped entirely. With 'all', content for any matching file or dataset will be dropped entirely. [Default: 'filecontent']

- **reckless** (*{'modification', 'availability', 'undead', 'kill', None}, optional*) – disable individual or all data safety measures that would normally prevent potentially irreversible data-loss. With 'modification', unsaved modifications in a dataset will not be detected. This improves performance at the cost of permitting potential loss of unsaved or untracked dataset components. With 'availability', detection of dataset/branch-states that are only available in the local dataset, and detection of an insufficient number of file- content copies will be disabled. Especially the latter is a potentially expensive check which might involve numerous network transactions. With 'undead', detection of whether a to-be-removed local annex is still known to exist in the network of dataset-clones is disabled. This could cause zombie-records of invalid file availability. With 'kill', all safety-checks are disabled. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform drop from. If no dataset is given, the current working directory is used as operation context. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **check** (*bool, optional*) – DEPRECATED: use '–reckless availability'. [Default: None]
- **if_dirty** – DEPRECATED and IGNORED: use –reckless instead. [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.api.get

`datalad.api.get`(*path=None*, *, *source=None*, *dataset=None*, *recursive=False*, *recursion_limit=None*, *get_data=True*, *description=None*, *reckless=None*, *jobs='auto'*)

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

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

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

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

Getting subdatasets

Just as DataLad supports getting file content from more than one location, the same is supported for subdatasets, including a ranking of individual sources for prioritization.

The following location candidates are considered. For each candidate a cost is given in parenthesis, higher values indicate higher cost, and thus lower priority:

- A datalad URL recorded in *.gitmodules* (cost 590). This allows for datalad URLs that require additional handling/resolution by datalad, like ria-schemes (ria+http, ria+ssh, etc.)
- A URL or absolute path recorded for git in *.gitmodules* (cost 600).
- URL of any configured superdataset remote that is known to have the desired submodule commit, with the submodule path appended to it. There can be more than one candidate (cost 650).
- In case *.gitmodules* contains a relative path instead of a URL, the URL of any configured superdataset remote that is known to have the desired submodule commit, with this relative path appended to it. There can be more than one candidate (cost 650).
- In case *.gitmodules* contains a relative path as a URL, the absolute path of the superdataset, appended with this relative path (cost 900).

Additional candidate URLs can be generated based on templates specified as configuration variables with the pattern

datalad.get.subdataset-source-candidate-<name>

where *name* is an arbitrary identifier. If *name* starts with three digits (e.g. '400myserver') these will be interpreted as a cost, and the respective candidate will be sorted into the generated candidate list according to this cost. If no cost is given, a default of 700 is used.

A template string assigned to such a variable can utilize the Python format mini language and may reference a number of properties that are inferred from the parent dataset's knowledge about the target subdataset. Properties include any submodule property specified in the respective *.gitmodules* record. For convenience, an existing *datalad-id* record is made available under the shortened name *id*.

Additionally, the URL of any configured remote that contains the respective submodule commit is available as *remoteurl-<name>* property, where *name* is the configured remote name.

Hence, such a template could be *http://example.org/datasets/{id}* or *http://example.org/datasets/{path}*, where *{id}* and *{path}* would be replaced by the *datalad-id* or *path* entry in the *.gitmodules* record.

If this config is committed in *.datalad/config*, a clone of a dataset can look up any subdataset's URL according to such scheme(s) irrespective of what URL is recorded in *.gitmodules*.

Lastly, all candidates are sorted according to their cost (lower values first), and duplicate URLs are stripped, while preserving the first item in the candidate list.

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

Examples

Get a single file:

```
> get('path/to/file')
```

Get contents of a directory:

```
> get('path/to/dir/')
```

Get all contents of the current dataset and its subdatasets:

```
> get(dataset='.', recursive=True)
```

Get (clone) a registered subdataset, but don't retrieve data:

```
> get('path/to/subds', get_data=False)
```

Parameters

- **path** (*sequence of str or None, optional*) – path/name of the requested dataset component. The component must already be known to a dataset. To add new components to a dataset use the *add* command. [Default: None]
- **source** (*str or None, optional*) – label of the data source to be used to fulfill requests. This can be the name of a dataset sibling or another known source. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the add operation on, in which case *path* arguments are interpreted as being relative to this dataset. If no dataset is given, an attempt is made to identify a dataset for each input *path*. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]

- **recursion_limit** (*int or {'existing'} or None, optional*) – limit recursion into subdataset to the given number of levels. Alternatively, 'existing' will limit recursion to subdatasets that already existed on the filesystem at the start of processing, and prevent new subdatasets from being obtained recursively. [Default: None]
- **get_data** (*bool, optional*) – whether to obtain data for all file handles. If disabled, *get* operations are limited to dataset handles. [Default: True]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. [Default: None]
- **reckless** (*{None, True, False, 'auto', 'ephemeral'} or shared-..., optional*) – Obtain a dataset or subdatset and set it up in a potentially unsafe way for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git-annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. [Default: 'auto']
- **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 re-

port 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.api.install

datalad.api.**install**(*path=None*, ***, *source=None*, *dataset=None*, *get_data=False*, *description=None*, *recursive=False*, *recursion_limit=None*, *reckless=None*, *jobs='auto'*, *branch=None*)

Install one or many datasets from remote URL(s) or local PATH source(s).

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

If no explicit *source* option is specified, then all positional URL- OR-PATH arguments are considered to be "sources" if they are URLs or target locations if they are paths. If a target location path corresponds to a submodule, the source location for it is figured out from its record in the *.gitmodules*. If *source* is specified, then a single optional positional PATH would be taken as the destination path for that dataset.

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

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

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

Examples

Install a dataset from GitHub into the current directory:

```
> install(source='https://github.com/datalad-datasets/longnow-podcasts.git')
```

Install a dataset as a subdataset into the current dataset:

```
> install(dataset='.',
          source='https://github.com/datalad-datasets/longnow-podcasts.git')
```

Install a dataset into 'podcasts' (not 'longnow-podcasts') directory, and get all content right away:

```
> install(path='podcasts',
          source='https://github.com/datalad-datasets/longnow-podcasts.git',
          get_data=True)
```

Install a dataset with all its subdatasets:

```
> install(source='https://github.com/datalad-datasets/longnow-podcasts.git',
          recursive=True)
```

Parameters

- **path** – path/name of the installation target. If no *path* is provided a destination path will be derived from a source URL similar to git clone. [Default: None]
- **source** (*str or None, optional*) – URL or local path of the installation source. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the install operation on. If no dataset is given, an attempt is made to identify the dataset in a parent directory of the current working directory and/or the *path* given. [Default: None]
- **get_data** (*bool, optional*) – if given, obtain all data content too. [Default: False]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **reckless** (*{None, True, False, 'auto', 'ephemeral'} or shared-..., optional*) – Obtain a dataset or subdatset and set it up in a potentially unsafe way for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git-annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. [Default: 'auto']
- **branch** (*str or None, optional*) – Clone source at this branch or tag. This option applies only to the top-level dataset not any subdatasets that may be cloned when installing

recursively. Note that if the source is a RIA URL with a version, it takes precedence over this option. [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: <function is_result_matching_pathsource_argument at 0x7fc7b1826ca0>]
- **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: 'successdatasets-or- 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: 'item-or-list']

datalad.api.push

datalad.api.**push**(*path=None*, *, *dataset=None*, *to=None*, *since=None*, *data='auto-if-wanted'*, *force=None*, *recursive=False*, *recursion_limit=None*, *jobs=None*)

Push a dataset to a known sibling.

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

By default, all files tracked in the last saved state (of the current branch) will be copied to the target location. Optionally, it is possible to limit a push to changes relative to a particular point in the version history of a dataset (e.g. a release tag) using the since option in conjunction with the specification of a reference dataset. In recursive

mode subdatasets will also be evaluated, and only those subdatasets are pushed where a change was recorded that is reflected in the current state of the top-level reference dataset.

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

Parameters

- **path** (*sequence of str or None, optional*) – path to constrain a push to. If given, only data or changes for those paths are considered for a push. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to push. [Default: None]
- **to** (*str or None, optional*) – name of the target sibling. If no name is given an attempt is made to identify the target based on the dataset's configuration (i.e. a configured tracking branch, or a single sibling that is configured for push). [Default: None]
- **since** (*str or None, optional*) – specifies commit-ish (tag, shasum, etc.) from which to look for changes to decide whether pushing is necessary. If '^' is given, the last state of the current branch at the sibling is taken as a starting point. [Default: None]
- **data** (*{'anything', 'nothing', 'auto', 'auto-if-wanted'}, optional*) – what to do with (annex'ed) data. 'anything' would cause transfer of all annexed content, 'nothing' would avoid call to *git annex copy* altogether. 'auto' would use 'git annex copy' with '–auto' thus transferring only data which would satisfy "wanted" or "numcopies" settings for the remote (thus "nothing" otherwise). 'auto-if-wanted' would enable '–auto' mode only if there is a "wanted" setting for the remote, and transfer 'anything' otherwise. [Default: 'auto-if-wanted']
- **force** (*{'all', 'gitpush', 'checkdatapresent', None}, optional*) – force particular operations, possibly overruling safety protections or optimizations: use –force with git-push ('gitpush'); do not use –fast with git-annex copy ('checkdatapresent'); combine all force modes ('all'). [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item. [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.api.remove

datalad.api.**remove**(*path=None*, *, *dataset=None*, *drop='datasets'*, *reckless=None*, *message=None*, *jobs=None*, *recursive=None*, *check=None*, *save=None*, *if_dirty=None*)

Remove components from datasets

Removing "unlinks" a dataset component, such as a file or subdataset, from a dataset. Such a removal advances the state of a dataset, just like adding new content. A remove operation can be undone, by restoring a previous dataset state, but might require re-obtaining file content and subdatasets from remote locations.

This command relies on the 'drop' command for safe operation. By default, only file content from datasets which will be uninstalled as part of a removal will be dropped. Otherwise file content is retained, such that restoring a previous version also immediately restores file content access, just as it is the case for files directly committed to Git. This default behavior can be changed to always drop content prior removal, for cases where a minimal storage footprint for local datasets installations is desirable.

Removing a dataset component is always a recursive operation. Removing a directory, removes all content underneath the directory too. If subdatasets are located under a to-be-removed path, they will be uninstalled entirely, and all their content dropped. If any subdataset can not be uninstalled safely, the remove operation will fail and halt.

Changed in version 0.16: More in-depth and comprehensive safety-checks are now performed by default. The `if_dirty` argument is ignored, will be removed in a future release, and can be removed for a safe-by-default behavior. For other cases consider the `reckless` argument. The `save` argument is ignored and will be removed in a future release, a dataset modification is now always saved. Consider save's `amend` argument for post-remove fix-ups. The `recursive` argument is ignored, and will be removed in a future release. Removal operations are always recursive, and the parameter can be stripped from calls for a safe-by-default behavior.

Deprecated since version 0.16: The `check` argument will be removed in a future release. It needs to be replaced with `reckless`.

Examples

Permanently remove a subdataset (and all further subdatasets contained in it) from a dataset:

```
> remove(dataset='path/to/dataset', path='path/to/subds')
```

Permanently remove a superdataset (with all subdatasets) from the filesystem:

```
> remove(dataset='path/to/dataset')
```

DANGER-ZONE: Fast wipe-out a dataset and all its subdataset, bypassing all safety checks:

```
> remove(dataset='path/to/dataset', reckless='kill')
```

Parameters

- **path** (*sequence of str or None, optional*) – path of a dataset or dataset component to be removed. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to perform remove from. If no dataset is given, the current working directory is used as operation context. [Default: None]
- **drop** (*{'datasets', 'all'}, optional*) – which dataset components to drop prior removal. This parameter is passed on to the underlying drop operation as its 'what' argument. [Default: 'datasets']
- **reckless** (*{'modification', 'availability', 'undead', 'kill', None}, optional*) – disable individual or all data safety measures that would normally prevent potentially irreversible data-loss. With 'modification', unsaved modifications in a dataset will not be detected. This improves performance at the cost of permitting potential loss of unsaved or untracked dataset components. With 'availability', detection of dataset/branch-states that are only available in the local dataset, and detection of an insufficient number of file- content copies will be disabled. Especially the latter is a potentially expensive check which might involve numerous network transactions. With 'undead', detection of whether a to-be-removed local annex is still known to exist in the network of dataset-clones is disabled. This could cause zombie-records of invalid file availability. With 'kill', all safety-checks are disabled. [Default: None]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **recursive** – DEPRECATED and IGNORED: removal is always a recursive operation. [Default: None]
- **check** (*bool, optional*) – DEPRECATED: use '–reckless availability'. [Default: None]
- **save** (*bool, optional*) – DEPRECATED and IGNORED; use *save –amend* instead. [Default: None]
- **if_dirty** – DEPRECATED and IGNORED: use –reckless instead. [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.api.save

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

Save the current state of a dataset

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

Note: Before Git v2.22, any Git repository without an initial commit located inside a Dataset is ignored, and content underneath it will be saved to the respective superdataset. DataLad datasets always have an initial commit, hence are not affected by this behavior.

Examples

Save any content underneath the current directory, without altering any potential subdataset:

```
> save(path='.')
```

Save specific content in the dataset:

```
> save(path='myfile.txt')
```

Attach a commit message to save:

```
> save(path='myfile.txt', message='add file')
```

Save any content underneath the current directory, and recurse into any potential subdatasets:

```
> save(path='.', recursive=True)
```

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

```
> save(path='.', updated=True)
```

Tag the most recent saved state of a dataset:

```
> save(version_tag='bestyet')
```

Save a specific change but integrate into last commit keeping the already recorded commit message:

```
> save(path='myfile.txt', amend=True)
```

Parameters

- **path** (*sequence of str or None, optional*) – path/name of the dataset component to save. If given, only changes made to those components are recorded in the new state. [Default: None]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to save. [Default: None]
- **version_tag** (*str or None, optional*) – an additional marker for that state. Every dataset that is touched will receive the tag. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **updated** (*bool, optional*) – if given, only saves previously tracked paths. [Default: False]
- **message_file** (*str or None, optional*) – take the commit message from this file. This flag is mutually exclusive with -m. [Default: None]
- **to_git** (*bool, optional*) – flag whether to add data directly to Git, instead of tracking data identity only. Use with caution, there is no guarantee that a file put directly into Git like this will not be annexed in a subsequent save operation. If not specified, it will be up to

git-annex to decide how a file is tracked, based on a dataset's configuration to track particular paths, file types, or file sizes with either Git or git-annex. (see https://git-annex.branchable.com/tips/largefiles). [Default: None]

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

datalad.api.status

datalad.api.**status**(*path=None*, *, *dataset=None*, *annex=None*, *untracked='normal'*, *recursive=False*, *recursion_limit=None*, *eval_subdataset_state='full'*, *report_filetype=None*)

Report on the state of dataset content.

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

Path conventions

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

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

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

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

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

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

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

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

Content types

The following content types are distinguished:

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

Content states

The following content states are distinguished:

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

Examples

Report on the state of a dataset:

```
> status()
```

Report on the state of a dataset and all subdatasets:

```
> status(recursive=True)
```

Address a subdataset record in a superdataset without causing a status query for the state _within_ the subdataset itself:

```
> status(dataset='.', path='mysubdataset')
```

Get a status query for the state within the subdataset without causing a status query for the superdataset (using trailing path separator in the query path)::

```
> status(dataset='.', path='mysubdataset/')
```

Report on the state of a subdataset in a superdataset and on the state within the subdataset:

```
> status(dataset='.', path=['mysubdataset', 'mysubdataset/'])
```

Report the file size of annexed content in a dataset:

```
> status(annex=True)
```

Parameters

- **path** (*sequence of str or None, optional*) – path to be evaluated. [Default: None]
- **dataset** (Dataset *or None, optional*) – specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **annex** (*{None, 'basic', 'availability', 'all'}, optional*) – Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). [Default: None]
- **untracked** (*{'no', 'normal', 'all'}, optional*) – If and how untracked content is reported when comparing a revision to the state of the working tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. [Default: 'normal']
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **eval_subdataset_state** (*{'no', 'commit', 'full'}, optional*) – Evaluation of subdataset state (clean vs. modified) can be expensive for deep dataset hierarchies as subdataset have to be tested recursively for uncommitted modifications. Setting this option to 'no' or

'commit' can substantially boost performance by limiting what is being tested. With 'no' no state is evaluated and subdataset result records typically do not contain a 'state' property. With 'commit' only a discrepancy of the HEAD commit shasum of a subdataset and the shasum recorded in the superdataset's record is evaluated, and the 'state' result property only reflects this aspect. With 'full' any other modification is considered too (see the 'untracked' option for further tailoring modification testing). [Default: 'full']

- **report_filetype** (*{'raw', 'eval', None}, optional*) – THIS OPTION IS IGNORED. It will be removed in a future release. Dataset component types are always reported as-is (previous 'raw' mode), unless annex-reporting is enabled with the *annex* option, in which case symlinks that represent annexed files will be reported as type='file'. [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.api.update

datalad.api.**update**(*path=None*, **, sibling=None*, *merge=False*, *how=None*, *how_subds=None*, *follow='sibling'*, *dataset=None*, *recursive=False*, *recursion_limit=None*, *fetch_all=None*, *reobtain_data=False*)

Update a dataset from a sibling.

Examples

Update from a particular sibling:

```
> update(sibling='siblingname')
```

Update from a particular sibling and merge the changes from a configured or matching branch from the sibling (see *follow* for details):

```
> update(sibling='siblingname', how='merge')
```

Update from the sibling 'origin', traversing into subdatasets. For subdatasets, merge the revision registered in the parent dataset into the current branch:

```
> update(sibling='origin', how='merge', follow='parentds', recursive=True)
```

Fetch and merge the remote tracking branch into the current dataset. Then update each subdataset by resetting its current branch to the revision registered in the parent dataset, fetching only if the revision isn't already present:

```
> update(how='merge', how_subds='reset', follow='parentds-lazy', recursive=True)
```

Parameters

- **path** (*sequence of str or None, optional*) – constrain to-be-updated subdatasets to the given path for recursive operation. [Default: None]
- **sibling** (*str or None, optional*) – name of the sibling to update from. When unspecified, updates from all siblings are fetched. If there is more than one sibling and changes will be brought into the working tree (as requested via *merge*, *how*, or *how_subds*), a sibling will be chosen based on the configured remote for the current branch. [Default: None]
- **merge** (*bool or {'any', 'ff-only'}, optional*) – merge obtained changes from the sibling. This is a subset of the functionality that can be achieved via the newer *how*. merge=True or merge="any" is equivalent to how="merge". merge="ff-only" is equivalent to how="ff-only". [Default: False]
- **how** (*{'fetch', 'merge', 'ff-only', 'reset', 'checkout', None}, optional*) – how to update the dataset. The default ("fetch") simply fetches the changes from the sibling but doesn't incorporate them into the working tree. A value of "merge" or "ff-only" merges in changes, with the latter restricting the allowed merges to fast-forwards. "reset" incorporates the changes with 'git reset –hard <target>', staying on the current branch but discarding any changes that aren't shared with the target. "checkout", on the other hand, runs 'git checkout <target>', switching from the current branch to a detached state. When recursive=True is specified, this action will also apply to subdatasets unless overridden by *how_subds*. [Default: None]
- **how_subds** (*{'fetch', 'merge', 'ff-only', 'reset', 'checkout', None}, optional*) – Override the behavior of *how* in subdatasets. [Default: None]

- **follow** (*{'sibling', 'parentds', 'parentds-lazy'}, optional*) – source of updates for subdatasets. For 'sibling', the update will be done by merging in a branch from the (specified or inferred) sibling. The branch brought in will either be the current branch's configured branch, if it points to a branch that belongs to the sibling, or a sibling branch with a name that matches the current branch. For 'parentds', the revision registered in the parent dataset of the subdataset is merged in. 'parentds-lazy' is like 'parentds', but prevents fetching from a subdataset's sibling if the registered revision is present in the subdataset. Note that the current dataset is always updated according to 'sibling'. This option has no effect unless a merge is requested and recursive=True is specified. [Default: 'sibling']
- **dataset** (*Dataset or None, optional*) – specify the dataset to update. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **fetch_all** (*bool, optional*) – this option has no effect and will be removed in a future version. When no siblings are given, an all-sibling update will be performed. [Default: None]
- **reobtain_data** (*bool, optional*) – if enabled, file content that was present before an update will be re-obtained in case a file was changed by the update. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result

transformation can be given. [Default: None]

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

datalad.api.unlock

datalad.api.**unlock**(*path=None*, *, *dataset=None*, *recursive=False*, *recursion_limit=None*)

Unlock file(s) of a dataset

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

Examples

Unlock a single file:

```
> unlock(path='path/to/file')
```

Unlock all contents in the dataset:

```
> unlock('.')
```

Parameters

- **path** (*sequence of str or None, optional*) – file(s) to unlock. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to unlock files in. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]
- **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']

Reproducible execution

api.run([cmd, dataset, inputs, outputs, ...])	Run an arbitrary shell command and record its impact on a dataset.
api.rerun([revision, since, dataset, ...])	Re-execute previous *datalad run* commands.
api.run_procedure([spec, dataset, discover, ...])	Run prepared procedures (DataLad scripts) on a dataset

datalad.api.run

datalad.api.**run**(*cmd=None*, ***, *dataset=None*, *inputs=None*, *outputs=None*, *expand=None*, *assume_ready=None*, *explicit=False*, *message=None*, *sidecar=None*, *dry_run=None*, *jobs=None*)

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

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

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

If the given command errors, a *CommandError* exception with the same exit code will be raised, and no modifications will be saved. A command execution will not be attempted, by default, when an error occurred during input or output preparation. This default `stop` behavior can be overridden via *on_failure=*. . . .

In the presence of subdatasets, the full dataset hierarchy will be checked for unsaved changes prior command execution, and changes in any dataset will be saved after execution. Any modification of subdatasets is also saved in their respective superdatasets to capture a comprehensive record of the entire dataset hierarchy state. The associated provenance record is duplicated in each modified (sub)dataset, although only being fully interpretable and re-executable in the actual top-level superdataset. For this reason the provenance record contains the dataset ID of that superdataset.

Command format

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

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

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

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

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

```
% datalad configuration --scope branch set datalad.run.substitutions.
↪name=joe
% datalad save -m "Configure name placeholder" .datalad/config
```

Access the new placeholder in a command:

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

Examples

Run an executable script and record the impact on a dataset:

```
> run(message='run my script', cmd='code/script.sh')
```

Run a command and specify a directory as a dependency for the run. The contents of the dependency will be retrieved prior to running the script:

```
> run(cmd='code/script.sh', message='run my script',
      inputs=['data/*'])
```

Run an executable script and specify output files of the script to be unlocked prior to running the script:

```
> run(cmd='code/script.sh', message='run my script',
      inputs=['data/*'], outputs=['output_dir'])
```

Specify multiple inputs and outputs:

```
> run(cmd='code/script.sh',
      message='run my script',
      inputs=['data/*', 'datafile.txt'],
      outputs=['output_dir', 'outfile.txt'])
```

Use ** to match any file at any directory depth recursively. Single * does not check files within matched directories.:

```
> run(cmd='code/script.sh',
      message='run my script',
      inputs=['data/**/*.dat'],
      outputs=['output_dir/**'])
```

Parameters

- **cmd** – command for execution. A leading '–' can be used to disambiguate this command from the preceding options to DataLad. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to record the command results in. An attempt is made to identify the dataset based on the current working directory. If a dataset is given, the command will be executed in the root directory of this dataset. [Default: None]
- **inputs** – A dependency for the run. Before running the command, the content for this relative path will be retrieved. A value of "." means "run datalad get .". The value can also be a glob. [Default: None]
- **outputs** – Prepare this relative path to be an output file of the command. A value of "." means "run datalad unlock ." (and will fail if some content isn't present). For any other value, if the content of this file is present, unlock the file. Otherwise, remove it. The value can also be a glob. [Default: None]
- **expand** (*{None, 'inputs', 'outputs', 'both'}, optional*) – Expand globs when storing inputs and/or outputs in the commit message. [Default: None]
- **assume_ready** (*{None, 'inputs', 'outputs', 'both'}, optional*) – Assume that inputs do not need to be retrieved and/or outputs do not need to unlocked or removed before running the command. This option allows you to avoid the expense of these preparation steps if you know that they are unnecessary. [Default: None]
- **explicit** (*bool, optional*) – Consider the specification of inputs and outputs to be explicit. Don't warn if the repository is dirty, and only save modifications to the listed outputs. [Default: False]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [Default: None]
- **sidecar** (*None or bool, optional*) – By default, the configuration variable 'datalad.run.record-sidecar' determines whether a record with information on a command's execution is placed into a separate record file instead of the commit message (default: off). This option can be used to override the configured behavior on a case-by-case basis. Sidecar files are placed into the dataset's '.datalad/runinfo' directory (customizable via the 'datalad.run.record-directory' configuration variable). [Default: None]
- **dry_run** (*{None, 'basic', 'command'}, optional*) – Do not run the command; just display details about the command execution. A value of "basic" reports a few important details about the execution, including the expanded command and expanded inputs and outputs. "command" displays the expanded command only. Note that input and output globs underneath an uninstalled dataset will be left unexpanded because no subdatasets will be installed for a dry run. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]

- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'stop']
- **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.api.rerun

datalad.api.**rerun**(*revision=None*, *, *since=None*, *dataset=None*, *branch=None*, *message=None*, *onto=None*, *script=None*, *report=False*, *assume_ready=None*, *explicit=False*, *jobs=None*)

Re-execute previous *datalad run* commands.

This will unlock any dataset content that is on record to have been modified by the command in the specified revision. It will then re-execute the command in the recorded path (if it was inside the dataset). Afterwards, all modifications will be saved.

Report mode

When called with report=True, this command reports information about what would be re-executed as a series of records. There will be a record for each revision in the specified revision range. Each of these will have one of the following "rerun_action" values:

- run: the revision has a recorded command that would be re-executed

- skip-or-pick: the revision does not have a recorded command and would be either skipped or cherry picked
- merge: the revision is a merge commit and a corresponding merge would be made

The decision to skip rather than cherry pick a revision is based on whether the revision would be reachable from HEAD at the time of execution.

In addition, when a starting point other than HEAD is specified, there is a rerun_action value "checkout", in which case the record includes information about the revision the would be checked out before rerunning any commands.

Note: Currently the "onto" feature only sets the working tree of the current dataset to a previous state. The working trees of any subdatasets remain unchanged.

Examples

Re-execute the command from the previous commit:

```
> rerun()
```

Re-execute any commands in the last five commits:

```
> rerun(since='HEAD~5')
```

Do the same as above, but re-execute the commands on top of HEAD~5 in a detached state:

```
> rerun(onto='', since='HEAD~5')
```

Parameters

- **revision** (*str or None, optional*) – rerun command(s) in *revision*. By default, the command from this commit will be executed, but *since* can be used to construct a revision range. The default value is like "HEAD" but resolves to the main branch when on an adjusted branch. [Default: None]
- **since** (*str or None, optional*) – If *since* is a commit-ish, the commands from all commits that are reachable from *revision* but not *since* will be re-executed (in other words, the commands in git log SINCE..REVISION). If SINCE is an empty string, it is set to the parent of the first commit that contains a recorded command (i.e., all commands in git log REVISION will be re-executed). [Default: None]
- **dataset** (Dataset *or None, optional*) – specify the dataset from which to rerun a recorded command. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. If a dataset is given, the command will be executed in the root directory of this dataset. [Default: None]
- **branch** (*str or None, optional*) – create and checkout this branch before rerunning the commands. [Default: None]
- **message** (*str or None, optional*) – use MESSAGE for the reran commit rather than the recorded commit message. In the case of a multi-commit rerun, all the reran commits will have this message. [Default: None]
- **onto** (*str or None, optional*) – start point for rerunning the commands. If not specified, commands are executed at HEAD. This option can be used to specify an alternative start point, which will be checked out with the branch name specified by *branch* or in a detached

state otherwise. As a special case, an empty value for this option means the parent of the first run commit in the specified revision list. [Default: None]

- **script** (*str or None, optional*) – extract the commands into this file rather than re-running. Use - to write to stdout instead. [Default: None]
- **report** (*bool, optional*) – Don't actually re-execute anything, just display what would be done. [Default: False]
- **assume_ready** (*{None, 'inputs', 'outputs', 'both'}, optional*) – Assume that inputs do not need to be retrieved and/or outputs do not need to unlocked or removed before running the command. This option allows you to avoid the expense of these preparation steps if you know that they are unnecessary. Note that this option also affects any additional outputs that are automatically inferred based on inspecting changed files in the run commit. [Default: None]
- **explicit** (*bool, optional*) – Consider the specification of inputs and outputs in the run record to be explicit. Don't warn if the repository is dirty, and only save modifications to the outputs from the original record. Note that when several run commits are specified, this applies to every one. Care should also be taken when using *onto* because checking out a new HEAD can easily fail when the working tree has modifications. [Default: False]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide

the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]

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

datalad.api.run_procedure

datalad.api.**run_procedure**(*spec=None*, *, *dataset=None*, *discover=False*, *help_proc=False*)

Run prepared procedures (DataLad scripts) on a dataset

Concept

A "procedure" is an algorithm with the purpose to process a dataset in a particular way. Procedures can be useful in a wide range of scenarios, like adjusting dataset configuration in a uniform fashion, populating a dataset with particular content, or automating other routine tasks, such as synchronizing dataset content with certain siblings.

Implementations of some procedures are shipped together with DataLad, but additional procedures can be provided by 1) any DataLad extension, 2) any (sub-)dataset, 3) a local user, or 4) a local system administrator. DataLad will look for procedures in the following locations and order:

Directories identified by the configuration settings

- 'datalad.locations.user-procedures' (determined by platformdirs.user_config_dir; defaults to '$HOME/.config/datalad/procedures' on GNU/Linux systems)
- 'datalad.locations.system-procedures' (determined by platformdirs.site_config_dir; defaults to '/etc/xdg/datalad/procedures' on GNU/Linux systems)
- 'datalad.locations.dataset-procedures'

and subsequently in the 'resources/procedures/' directories of any installed extension, and, lastly, of the DataLad installation itself.

Please note that a dataset that defines 'datalad.locations.dataset-procedures' provides its procedures to any dataset it is a subdataset of. That way you can have a collection of such procedures in a dedicated dataset and install it as a subdataset into any dataset you want to use those procedures with. In case of a naming conflict with such a dataset hierarchy, the dataset you're calling run-procedures on will take precedence over its subdatasets and so on.

Each configuration setting can occur multiple times to indicate multiple directories to be searched. If a procedure matching a given name is found (filename without a possible extension), the search is aborted and this implementation will be executed. This makes it possible for individual datasets, users, or machines to override externally provided procedures (enabling the implementation of customizable processing "hooks").

Procedure implementation

A procedure can be any executable. Executables must have the appropriate permissions and, in the case of a script, must contain an appropriate "shebang" line. If a procedure is not executable, but its filename ends with '.py', it is automatically executed by the 'python' interpreter (whichever version is available in the present environment). Likewise, procedure implementations ending on '.sh' are executed via 'bash'.

Procedures can implement any argument handling, but must be capable of taking at least one positional argument (the absolute path to the dataset they shall operate on).

For further customization there are two configuration settings per procedure available:

- 'datalad.procedures.<NAME>.call-format' fully customizable format string to determine how to execute procedure NAME (see also datalad-run). It currently requires to include the following placeholders:

  - '{script}': will be replaced by the path to the procedure
  - '{ds}': will be replaced by the absolute path to the dataset the procedure shall operate on
  - '{args}': (not actually required) will be replaced by

    all but the first element of *spec* if *spec* is a list or tuple As an example the default format string for a call to a python script is: "python {script} {ds} {args}"

- 'datalad.procedures.<NAME>.help' will be shown on *datalad run-procedure –help-proc NAME* to provide a description and/or usage info for procedure NAME

Examples

Find out which procedures are available on the current system:

```
> run_procedure(discover=True)
```

Run the 'yoda' procedure in the current dataset:

```
> run_procedure(spec='cfg_yoda', recursive=True)
```

Parameters

- **spec** – Name and possibly additional arguments of the to-be-executed procedure. [PY: Can also be a dictionary coming from run- procedure(discover=True).]. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to run the procedure on. An attempt is made to identify the dataset based on the current working directory. [Default: None]
- **discover** (*bool, optional*) – if given, all configured paths are searched for procedures and one result record per discovered procedure is yielded, but no procedure is executed. [Default: False]
- **help_proc** (*bool, optional*) – if given, get a help message for procedure NAME from config setting datalad.procedures.NAME.help. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value

choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']

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

Plumbing commands

api.clean(*[, dataset, what, dry_run, ...])	Clean up after DataLad (possible temporary files etc.)
api.clone(source[, path, git_clone_opts, ...])	Obtain a dataset (copy) from a URL or local directory
api.copy_file([path, dataset, recursive, ...])	Copy files and their availability metadata from one dataset to another.
api.create_test_dataset([path, spec, seed])	Create test (meta-)dataset.
api.diff([path, fr, to, dataset, annex, ...])	Report differences between two states of a dataset (hierarchy)
api.download_url(urls, *[, dataset, path, ...])	Download content
api.foreach_dataset(cmd, *[, cmd_type, ...])	Run a command or Python code on the dataset and/or each of its sub-datasets.
api.siblings([action, dataset, name, url, ...])	Manage sibling configuration
api.sshrun(login, cmd, *[, port, ipv4, ...])	Run command on remote machines via SSH.
api.subdatasets([path, dataset, state, ...])	Report subdatasets and their properties.

datalad.api.clean

datalad.api.**clean**(*, *dataset=None*, *what=None*, *dry_run=False*, *recursive=False*, *recursion_limit=None*)

Clean up after DataLad (possible temporary files etc.)

Removes temporary files and directories left behind by DataLad and git-annex in a dataset.

Examples

Clean all known temporary locations of a dataset:

```
> clean()
```

Report on all existing temporary locations of a dataset:

```
> clean(dry_run=True)
```

Clean all known temporary locations of a dataset and all its subdatasets:

```
> clean(recursive=True)
```

Clean only the archive extraction caches of a dataset and all its subdatasets:

```
> clean(what='cached-archives', recursive=True)
```

Report on existing annex transfer files of a dataset and all its subdatasets:

```
> clean(what='annex-transfer', recursive=True, dry_run=True)
```

Parameters

- **dataset** (*Dataset or None, optional*) – specify the dataset to perform the clean operation on. If no dataset is given, an attempt is made to identify the dataset in current working directory. [Default: None]
- **what** (*sequence of {'cached-archives', 'annex-tmp', 'annex-transfer', 'search-index'} or None, optional*) – What to clean. If none specified – all known targets are considered. [Default: None]
- **dry_run** (*bool, optional*) – Report on cleanable locations - not actually cleaning up anything. [Default: False]
- **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; '<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.api.clone

datalad.api.**clone**(*source*, *path=None*, *git_clone_opts=None*, *, *dataset=None*, *description=None*, *reckless=None*)

Obtain a dataset (copy) from a URL or local directory

The purpose of this command is to obtain a new clone (copy) of a dataset and place it into a not-yet-existing or empty directory. As such *clone* provides a strict subset of the functionality offered by *install*. Only a single dataset can be obtained, and immediate recursive installation of subdatasets is not supported. However, once a (super)dataset is installed via *clone*, any content, including subdatasets can be obtained by a subsequent *get* command.

Primary differences over a direct *git clone* call are 1) the automatic initialization of a dataset annex (pure Git repositories are equally supported); 2) automatic registration of the newly obtained dataset as a subdataset (submodule), if a parent dataset is specified; 3) support for additional resource identifiers (DataLad resource identifiers as used on datasets.datalad.org, and RIA store URLs as used for store.datalad.org - optionally in specific versions as identified by a branch or a tag; see examples); and 4) automatic configurable generation of alternative access URL for common cases (such as appending '.git' to the URL in case the accessing the base URL failed).

In case the clone is registered as a subdataset, the original URL passed to *clone* is recorded in *.gitmodules* of the parent dataset in addition to the resolved URL used internally for git-clone. This allows to preserve datalad specific URLs like ria+ssh://... for subsequent calls to *get* if the subdataset was locally removed later on.

By default, the command returns a single Dataset instance for an installed dataset, regardless of whether it was newly installed ('ok' result), or found already installed from the specified source ('notneeded' result).

URL mapping configuration

'clone' supports the transformation of URLs via (multi-part) substitution specifications. A substitution specification is defined as a configuration setting 'datalad.clone.url-substition.<seriesID>' with a string containing a match and substitution expression, each following Python's regular expression syntax. Both expressions are concatenated to a single string with an arbitrary delimiter character. The delimiter is defined by prefixing the string with the delimiter. Prefix and delimiter are stripped from the expressions (Example: ",^http://(.*)$,https://1"). This setting can be defined multiple times, using the same '<seriesID>'. Substitutions in a series will be applied incrementally, in order of their definition. The first substitution in such a series must match, otherwise no further substitutions in a series will be considered. However, following the first match all further substitutions in a series are processed, regardless whether intermediate expressions match or not. Substitution series themselves have no particular order, each matching series will result in a candidate clone URL. Consequently, the initial match specification in a series should be as precise as possible to prevent inflation of candidate URLs.

See also:

handbook:3-001 (http://handbook.datalad.org/symbols)
More information on Remote Indexed Archive (RIA) stores

Examples

Install a dataset from GitHub into the current directory:

```
> clone(source='https://github.com/datalad-datasets/longnow-podcasts.git')
```

Install a dataset into a specific directory:

```
> clone(source='https://github.com/datalad-datasets/longnow-podcasts.git',
        path='myfavpodcasts')
```

Install a dataset as a subdataset into the current dataset:

```
> clone(dataset='.',
        source='https://github.com/datalad-datasets/longnow-podcasts.git')
```

Install the main superdataset from datasets.datalad.org:

```
> clone(source='///')
```

Install a dataset identified by a literal alias from store.datalad.org:

```
> clone(source='ria+http://store.datalad.org#~hcp-openaccess')
```

Install a dataset in a specific version as identified by a branch or tag name from store.datalad.org:

```
> clone(source='ria+http://store.datalad.org#76b6ca66-36b1-11ea-a2e6-
↪f0d5bf7b5561@myidentifier')
```

Install a dataset with group-write access permissions:

```
> clone(source='http://example.com/dataset', reckless='shared-group')
```

Parameters

- **source** (*str*) – URL, DataLad resource identifier, local path or instance of dataset to be cloned.
- **path** – path to clone into. If no *path* is provided a destination path will be derived from a source URL similar to git clone. [Default: None]
- **git_clone_opts** – A list of command line arguments to pass to git clone. Note that not all options will lead to viable results. For example '–single- branch' will not result in a functional annex repository because both a regular branch and the git-annex branch are required. Note that a version in a RIA URL takes precedence over '–branch'. [Default: None]
- **dataset** (*Dataset or None, optional*) – (parent) dataset to clone into. If given, the newly cloned dataset is registered as a subdataset of the parent. Also, if given, relative paths are interpreted as being relative to the parent dataset, and not relative to the working directory. [Default: None]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. [Default: None]
- **reckless** (*{None, True, False, 'auto', 'ephemeral'} or shared-..., optional*) – Obtain a dataset or subdatset and set it up in a potentially unsafe way

for performance, or access reasons. Use with care, any dataset is marked as 'untrusted'. The reckless mode is stored in a dataset's local configuration under 'datalad.clone.reckless', and will be inherited to any of its subdatasets. Supported modes are: ['auto']: hard-link files between local clones. In-place modification in any clone will alter original annex content. ['ephemeral']: symlink annex to origin's annex and discard local availability info via git-annex-dead 'here' and declares this annex private. Shares an annex between origin and clone w/o git-annex being aware of it. In case of a change in origin you need to update the clone before you're able to save new content on your end. Alternative to 'auto' when hardlinks are not an option, or number of consumed inodes needs to be minimized. Note that this mode can only be used with clones from non-bare repositories or a RIA store! Otherwise two different annex object tree structures (dirhashmixed vs dirhashlower) will be used simultaneously, and annex keys using the respective other structure will be inaccessible. ['shared-<mode>']: set up repository and annex permission to enable multi-user access. This disables the standard write protection of annex'ed files. <mode> can be any value support by 'git init –shared=', such as 'group', or 'all'. [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: constraint:action:{install}]
- **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: 'successdatasets-or- 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: 'item-or-list']

datalad.api.copy_file

datalad.api.**copy_file**(*path=None*, *, *dataset=None*, *recursive=False*, *target_dir=None*, *specs_from=None*, *message=None*)

Copy files and their availability metadata from one dataset to another.

The difference to a system copy command is that here additional content availability information, such as registered URLs, is also copied to the target dataset. Moreover, potentially required git-annex special remote configurations are detected in a source dataset and are applied to a target dataset in an analogous fashion. It is possible to copy a file for which no content is available locally, by just copying the required metadata on content identity and availability.

Note: At the moment, only URLs for the special remotes 'web' (git-annex built-in) and 'datalad' are recognized and transferred.

The interface is modeled after the POSIX 'cp' command, but with one additional way to specify what to copy where: *specs_from* allows the caller to flexibly input source-destination path pairs.

This command can copy files out of and into a hierarchy of nested datasets. Unlike with other DataLad command, the *recursive* switch does not enable recursion into subdatasets, but is analogous to the POSIX 'cp' command switch and enables subdirectory recursion, regardless of dataset boundaries. It is not necessary to enable recursion in order to save changes made to nested target subdatasets.

Examples

Copy a file into a dataset 'myds' using a path and a target directory specification, and save its addition to 'myds':

```
> copy_file('path/to/myfile', dataset='path/to/myds')
```

Copy a file to a dataset 'myds' and save it under a new name by providing two paths:

```
> copy_file(path=['path/to/myfile', 'path/to/myds/newname'],
            dataset='path/to/myds')
```

Copy a file into a dataset without saving it:

```
> copy_file('path/to/myfile', target_dir='path/to/myds/')
```

Copy a directory and its subdirectories into a dataset 'myds' and save the addition in 'myds':

```
> copy_file('path/to/dir/', recursive=True, dataset='path/to/myds')
```

Copy files using a path and optionally target specification from a file:

```
> copy_file(dataset='path/to/myds', specs_from='path/to/specfile')
```

Parameters

- **path** (*sequence of str or None, optional*) – paths to copy (and possibly a target path to copy to). [Default: None]
- **dataset** (*Dataset or None, optional*) – root dataset to save after copy operations are completed. All destination paths must be within this dataset, or its subdatasets. If no dataset is given, dataset modifications will be left unsaved. [Default: None]

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

datalad.api.create_test_dataset

datalad.api.**create_test_dataset**(*path=None*, *, *spec=None*, *seed=None*)

Create test (meta-)dataset.

Parameters

- **path** (*str or None, optional*) – path/name where to create (if specified, must not exist). [Default: None]
- **spec** (*str or None, optional*) – spec for hierarchy, defined as a min-max (min could be omitted to assume 0) defining how many (random number from min to max) of sub- datasets to generate at any given level of the hierarchy. Each level separated from each other with /. Example: 1-3/-2 would generate from 1 to 3 subdatasets at the top level, and up to two within those at the 2nd level. [Default: None]
- **seed** (*int or None, optional*) – seed for rng. [Default: None]

datalad.api.diff

datalad.api.**diff**(*path=None*, *, *fr='HEAD'*, *to=None*, *dataset=None*, *annex=None*, *untracked='normal'*, *recursive=False*, *recursion_limit=None*)

Report differences between two states of a dataset (hierarchy)

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

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

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

Examples

Show unsaved changes in a dataset:

```
> diff()
```

Compare a previous dataset state identified by shasum against current worktree:

```
> diff(fr='SHASUM')
```

Compare two branches against each other:

```
> diff(fr='branch1', to='branch2')
```

Show unsaved changes in the dataset and potential subdatasets:

```
> diff(recursive=True)
```

Show unsaved changes made to a particular file:

```
> diff(path='path/to/file')
```

Parameters

- **path** (*sequence of str or None, optional*) – path to constrain the report to. [Default: None]
- **fr** (*str, optional*) – original state to compare to, as given by any identifier that Git understands. [Default: 'HEAD']
- **to** (*str or None, optional*) – state to compare against the original state, as given by any identifier that Git understands. If none is specified, the state of the working tree will be compared. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **annex** (*{None, 'basic', 'availability', 'all'}, optional*) – Switch whether to include information on the annex content of individual files in the status report, such as recorded file size. By default no annex information is reported (faster). Three report modes are available: basic information like file size and key name ('basic'); additionally test whether file content is present in the local annex ('availability'; requires one or two additional file system stat calls, but does not call git-annex), this will add the result properties 'has_content' (boolean flag) and 'objloc' (absolute path to an existing annex object file); or 'all' which will report all available information (presently identical to 'availability'). [Default: None]
- **untracked** (*{'no', 'normal', 'all'}, optional*) – If and how untracked content is reported when comparing a revision to the state of the working tree. 'no': no untracked content is reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories. [Default: 'normal']
- **recursive** (*bool, optional*) – if set, recurse into potential 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; '<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.api.download_url

datalad.api.**download_url**(*urls*, *, *dataset=None*, *path=None*, *overwrite=False*, *archive=False*, *save=True*, *message=None*)

Download content

It allows for a uniform download interface to various supported URL schemes (see command help for details), re-using or asking for authentication details maintained by datalad.

Examples

Download files from an http and S3 URL:

```
> download_url(urls=['http://example.com/file.dat', 's3://bucket/file2.dat'])
```

Download a file to a path and provide a commit message:

```
> download_url(urls='s3://bucket/file2.dat', message='added a file', path='myfile.
↪dat')
```

Append a trailing slash to the target path to download into a specified directory:

```
> download_url(['http://example.com/file.dat'], path='data/')
```

Leave off the trailing slash to download into a regular file:

```
> download_url(['http://example.com/file.dat'], path='data')
```

Parameters

- **urls** (*non-empty sequence of str*) – URL(s) to be downloaded. Supported protocols: 'ftp', 'http', 'https', 's3', 'shub'.
- **dataset** (*Dataset or None, optional*) – specify the dataset to add files to. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. Use save=False to prevent adding files to the dataset. [Default: None]

- **path** (*str or None, optional*) – target for download. If the path has a trailing separator, it is treated as a directory, and each specified URL is downloaded under that directory to a base name taken from the URL. Without a trailing separator, the value specifies the name of the downloaded file (file name extensions inferred from the URL may be added to it, if they are not yet present) and only a single URL should be given. In both cases, leading directories will be created if needed. This argument defaults to the current directory. [Default: None]
- **overwrite** (*bool, optional*) – flag to overwrite it if target file exists. [Default: False]
- **archive** (*bool, optional*) – pass the downloaded files to add_archive_content(..., delete=True). [Default: False]
- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Giving this option will disable this behavior. [Default: True]
- **message** (*str or None, optional*) – a description of the state or the changes made to a dataset. [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.api.foreach_dataset

datalad.api.**foreach_dataset**(*cmd*, *, *cmd_type='auto'*, *dataset=None*, *state='present'*, *recursive=False*, *recursion_limit=None*, *contains=None*, *bottomup=False*, *subdatasets_only=False*, *output_streams='pass-through'*, *chpwd='ds'*, *safe_to_consume='auto'*, *jobs=None*)

Run a command or Python code on the dataset and/or each of its sub-datasets.

This command provides a convenience for the cases were no dedicated DataLad command is provided to operate across the hierarchy of datasets. It is very similar to *git submodule foreach* command with the following major differences

- by default (unless *subdatasets_only=True*) it would include operation on the original dataset as well,
- subdatasets could be traversed in bottom-up order,
- can execute commands in parallel (see *jobs* option), but would account for the order, e.g. in bottom-up order command is executed in super-dataset only after it is executed in all subdatasets.

Additional notes:

- for execution of "external" commands we use the environment used to execute external git and git-annex commands.

Command format

cmd_type='external': A few placeholders are supported in the command via Python format specification:

- "{pwd}" will be replaced with the full path of the current working directory.
- "{ds}" and "{refds}" will provide instances of the dataset currently operated on and the reference "context" dataset which was provided via `dataset` argument.
- "{tmpdir}" will be replaced with the full path of a temporary directory.

Examples

Aggressively git clean all datasets, running 5 parallel jobs:

```
> foreach_dataset(['git', 'clean', '-dfx'], recursive=True, jobs=5)
```

Parameters

- **cmd** – command for execution. For *cmd_type='exec'* or *cmd_type='eval'* (Python code) should be either a string or a list with only a single item. If 'eval', the actual function can be passed, which will be provided all placeholders as keyword arguments.
- **cmd_type** (*{'auto', 'external', 'exec', 'eval'}, optional*) – type of the command. *external*: to be run in a child process using dataset's runner; 'exec': Python source code to execute using 'exec(), no value returned; 'eval': Python source code to evaluate using 'eval()', return value is placed into 'result' field. 'auto': If used via Python API, and *cmd* is a Python function, it will use 'eval', and otherwise would assume 'external'. [Default: 'auto']
- **dataset** (*Dataset or None, optional*) – specify the dataset to operate on. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. [Default: None]
- **state** (*{'present', 'absent', 'any'}, optional*) – indicate which (sub)datasets to consider: either only locally present, absent, or any of those two kinds. [Default: 'present']

- **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]
- **contains** (*list of str or None, optional*) – limit to the subdatasets containing the given path. If a root path of a subdataset is given, the last considered dataset will be the subdataset itself. Can be a list with multiple paths, in which case datasets that contain any of the given paths will be considered. [Default: None]
- **bottomup** (*bool, optional*) – whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down. [Default: False]
- **subdatasets_only** (*bool, optional*) – whether to exclude top level dataset. It is implied if a non-empty *contains* is used. [Default: False]
- **output_streams** (*{'capture', 'pass-through', 'relpath'}, optional*) – ways to handle outputs. 'capture' and return outputs from 'cmd' in the record ('stdout', 'stderr'); 'pass-through' to the screen (and thus absent from returned record); prefix with 'relpath' captured output (similar to like grep does) and write to stdout and stderr. In 'relpath', relative path is relative to the top of the dataset if *dataset* is specified, and if not - relative to current directory. [Default: 'pass-through']
- **chpwd** (*{'ds', 'pwd'}, optional*) – 'ds' will change working directory to the top of the corresponding dataset. With 'pwd' no change of working directory will happen. Note that for Python commands, due to use of threads, we do not allow chdir=ds to be used with jobs > 1. Hint: use 'ds' and 'refds' objects' methods to execute commands in the context of those datasets. [Default: 'ds']
- **safe_to_consume** (*{'auto', 'all-subds-done', 'superds-done', 'always'}, optional*) – Important only in the case of parallel (jobs greater than 1) execution. 'all-subds-done' instructs to not consider superdataset until command finished execution in all subdatasets (it is the value in case of 'auto' if traversal is bottomup). 'superds-done' instructs to not process subdatasets until command finished in the super-dataset (it is the value in case of 'auto' in traversal is not bottom up, which is the default). With 'always' there is no constraint on either to execute in sub or super dataset. [Default: 'auto']
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and

an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']

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

datalad.api.siblings

datalad.api.**siblings**(*action='query', *, dataset=None, name=None, url=None, pushurl=None, description=None, fetch=False, as_common_datasrc=None, publish_depends=None, publish_by_default=None, annex_wanted=None, annex_required=None, annex_group=None, annex_groupwanted=None, inherit=False, get_annex_info=True, recursive=False, recursion_limit=None*)

Manage sibling configuration

This command offers four different actions: 'query', 'add', 'remove', 'configure', 'enable'. 'query' is the default action and can be used to obtain information about (all) known siblings. 'add' and 'configure' are highly similar actions, the only difference being that adding a sibling with a name that is already registered will fail, whereas re-configuring a (different) sibling under a known name will not be considered an error. 'enable' can be used to complete access configuration for non-Git sibling (aka git-annex special remotes). Lastly, the 'remove' action allows for the removal (or de-configuration) of a registered sibling.

For each sibling (added, configured, or queried) all known sibling properties are reported. This includes:

"name"
Name of the sibling

"path"
Absolute path of the dataset

"url"
For regular siblings at minimum a "fetch" URL, possibly also a "pushurl"

Additionally, any further configuration will also be reported using a key that matches that in the Git configuration.

By default, sibling information is rendered as one line per sibling following this scheme:

```
<dataset_path>: <sibling_name>(<+|->) [<access_specification]
```

where the + and - labels indicate the presence or absence of a remote data annex at a particular remote, and *access_specification* contains either a URL and/or a type label for the sibling.

Parameters

- **action** (*{'query', 'add', 'remove', 'configure', 'enable'}, optional*) – command action selection (see general documentation). [Default: 'query']
- **dataset** (*Dataset or None, optional*) – specify the dataset to configure. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. [Default: None]
- **name** (*str or None, optional*) – name of the sibling. For addition with path "URLs" and sibling removal this option is mandatory, otherwise the hostname part of a given URL is used as a default. This option can be used to limit 'query' to a specific sibling. [Default: None]
- **url** (*str or None, optional*) – the URL of or path to the dataset sibling named by *name*. For recursive operation it is required that a template string for building subdataset sibling URLs is given. List of currently available placeholders: %%NAME the name of the dataset, where slashes are replaced by dashes. [Default: None]
- **pushurl** (*str or None, optional*) – in case the *url* cannot be used to publish to the dataset sibling, this option specifies a URL to be used instead. If no *url* is given, *pushurl* serves as *url* as well. [Default: None]
- **description** (*str or None, optional*) – short description to use for a dataset location. Its primary purpose is to help humans to identify a dataset copy (e.g., "mike's dataset on lab server"). Note that when a dataset is published, this information becomes available on the remote side. [Default: None]
- **fetch** (*bool, optional*) – fetch the sibling after configuration. [Default: False]
- **as_common_datasrc** – configure a sibling as a common data source of the dataset that can be automatically used by all consumers of the dataset. The sibling must be a regular Git remote with a configured HTTP(S) URL. [Default: None]
- **publish_depends** (*list of str or None, optional*) – add a dependency such that the given existing sibling is always published prior to the new sibling. This equals setting a configuration item 'remote.SIBLINGNAME.datalad-publish-depends'. Multiple dependencies can be given as a list of sibling names. [Default: None]
- **publish_by_default** (*list of str or None, optional*) – add a refspec to be published to this sibling by default if nothing specified. [Default: None]
- **annex_wanted** (*str or None, optional*) – expression to specify 'wanted' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-wanted/ for more information. [Default: None]
- **annex_required** (*str or None, optional*) – expression to specify 'required' content for the repository/sibling. See https://git-annex.branchable.com/git-annex-required/ for more information. [Default: None]
- **annex_group** (*str or None, optional*) – expression to specify a group for the repository. See https://git- annex.branchable.com/git-annex-group/ for more information. [Default: None]
- **annex_groupwanted** (*str or None, optional*) – expression for the groupwanted. Makes sense only if annex_wanted="groupwanted" and annex-group is given too. See https://git-annex.branchable.com/git-annex-groupwanted/ for more information. [Default: None]
- **inherit** (*bool, optional*) – if sibling is missing, inherit settings (git config, git annex wanted/group/groupwanted) from its super-dataset. [Default: False]

- **get_annex_info** (*bool, optional*) – Whether to query all information about the annex configurations of siblings. Can be disabled if speed is a concern. [Default: True]
- **recursive** (*bool, optional*) – if set, recurse into potential 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; '<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.api.sshrun

datalad.api.**sshrun**(*login*, *cmd*, *, *port=None*, *ipv4=False*, *ipv6=False*, *options=None*, *no_stdin=False*)

Run command on remote machines via SSH.

This is a replacement for a small part of the functionality of SSH. In addition to SSH alone, this command can make use of datalad's SSH connection management. Its primary use case is to be used with Git as 'core.sshCommand' or via "GIT_SSH_COMMAND".

Configure *datalad.ssh.identityfile* to pass a file to the ssh's -i option.

Parameters

- **login** – [user@]hostname.
- **cmd** – command for remote execution.
- **port** – port to connect to on the remote host. [Default: None]
- **ipv4** (*bool, optional*) – use IPv4 addresses only. [Default: False]
- **ipv6** (*bool, optional*) – use IPv6 addresses only. [Default: False]
- **options** – configuration option passed to SSH. [Default: None]
- **no_stdin** (*bool, optional*) – Do not connect stdin to the process. [Default: False]

datalad.api.subdatasets

datalad.api.**subdatasets**(*path=None*, *, *dataset=None*, *state='any'*, *fulfilled=None(DEPRECATED)*, *recursive=False*, *recursion_limit=None*, *contains=None*, *bottomup=False*, *set_property=None*, *delete_property=None*)

Report subdatasets and their properties.

The following properties are reported (if possible) for each matching subdataset record.

"name"
Name of the subdataset in the parent (often identical with the relative path in the parent dataset)

"path"
Absolute path to the subdataset

"parentds"
Absolute path to the parent dataset

"gitshasum"
SHA1 of the subdataset commit recorded in the parent dataset

"state"
Condition of the subdataset: 'absent', 'present'

"gitmodule_url"
URL of the subdataset recorded in the parent

"gitmodule_name"
Name of the subdataset recorded in the parent

"gitmodule_<label>"
Any additional configuration property on record.

Performance note: Property modification, requesting *bottomup* reporting order, or a particular numerical *recursion_limit* implies an internal switch to an alternative query implementation for recursive query that is more flexible, but also notably slower (performs one call to Git per dataset versus a single call for all combined).

The following properties for subdatasets are recognized by DataLad (without the 'gitmodule_' prefix that is used in the query results):

"datalad-recursiveinstall"
If set to 'skip', the respective subdataset is skipped when DataLad is recursively installing its superdataset. However, the subdataset remains installable when explicitly requested, and no other features are impaired.

"datalad-url"
If a subdataset was originally established by cloning, 'datalad-url' records the URL that was used to do so. This might be different from 'url' if the URL contains datalad specific pieces like any URL of the form "ria+<some protocol>…".

Parameters

- **path** (*sequence of str or None, optional*) – path/name to query for subdatasets. Defaults to the current directory, or the entire dataset if called as a dataset method. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to query. If no dataset is given, an attempt is made to identify the dataset based on the input and/or the current working directory. [Default: None]
- **state** (*{'present', 'absent', 'any'}, optional*) – indicate which (sub)datasets to consider: either only locally present, absent, or any of those two kinds. [Default: 'any']
- **fulfilled** (*bool or None, optional*) – DEPRECATED: use *state* instead. If given, must be a boolean flag indicating whether to consider either only locally present or absent datasets. By default all subdatasets are considered regardless of their status. [Default: None(DEPRECATED)]
- **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]
- **contains** (*list of str or None, optional*) – limit to the subdatasets containing the given path. If a root path of a subdataset is given, the last considered dataset will be the subdataset itself. Can be a list with multiple paths, in which case datasets that contain any of the given paths will be considered. [Default: None]
- **bottomup** (*bool, optional*) – whether to report subdatasets in bottom-up order along each branch in the dataset tree, and not top-down. [Default: False]
- **set_property** (*list of 2-item sequence of str or None, optional*) – Name and value of one or more subdataset properties to be set in the parent dataset's .gitmodules file. The property name is case- insensitive, must start with a letter, and consist only of alphanumeric characters. The value can be a Python format() template string wrapped in '<>' (e.g. '<{gitmodule_name}>'). Supported keywords are any item reported in the result properties of this command, plus 'refds_relpath' and 'refds_relname': the relative path of a subdataset with respect to the base dataset of the command call, and, in the latter case, the same string with all directory separators replaced by dashes. [Default: None]
- **delete_property** (*list of str or None, optional*) – Name of one or more subdataset properties to be removed from the parent dataset's .gitmodules file. [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']

Miscellaneous commands

api.add_archive_content(archive, *[, ...])	Add content of an archive under git annex control.
api.add_readme([filename, dataset, existing])	Add basic information about DataLad datasets to a README file
api.addurls(urlfile, urlformat, ...[, ...])	Create and update a dataset from a list of URLs.
api.check_dates(paths, *[, reference_date, ...])	Find repository dates that are more recent than a reference date.
api.configuration([action, spec, scope, ...])	Get and set dataset, dataset-clone-local, or global configuration
api.export_archive([filename, dataset, ...])	Export the content of a dataset as a TAR/ZIP archive.
api.export_archive_ora(target[, opts, ...])	Export an archive of a local annex object store for the ORA remote.
api.export_to_figshare([filename, dataset, ...])	Export the content of a dataset as a ZIP archive to figshare
api.no_annex(dataset, pattern[, ref_dir, ...])	Configure a dataset to never put some content into the dataset's annex
api.shell_completion()	Display shell script for enabling shell completion for DataLad.
api.wtf(*[, dataset, sensitive, sections, ...])	Generate a report about the DataLad installation and configuration

datalad.api.add_archive_content

datalad.api.**add_archive_content**(*archive*, *, *dataset=None*, *annex=None*, *add_archive_leading_dir=False*, *strip_leading_dirs=False*, *leading_dirs_depth=None*, *leading_dirs_consider=None*, *use_current_dir=False*, *delete=False*, *key=False*, *exclude=None*, *rename=None*, *existing='fail'*, *annex_options=None*, *copy=False*, *commit=True*, *allow_dirty=False*, *stats=None*, *drop_after=False*, *delete_after=False*)

Add content of an archive under git annex control.

Given an already annex'ed archive, extract and add its files to the dataset, and reference the original archive as a custom special remote.

Examples

Add files from the archive 'big_tarball.tar.gz', but keep big_tarball.tar.gz in the index:

```
> add_archive_content(path='big_tarball.tar.gz')
```

Add files from the archive 'tarball.tar.gz', and remove big_tarball.tar.gz from the index:

```
> add_archive_content(path='big_tarball.tar.gz', delete=True)
```

Add files from the archive 's3.zip' but remove the leading directory:

```
> add_archive_content(path='s3.zip', strip_leading_dirs=True)
```

Parameters

- **archive** (*str*) – archive file or a key (if *key=True* specified).
- **dataset** (*Dataset or None, optional*) – "specify the dataset to save. [Default: None]
- **annex** – DEPRECATED. Use the 'dataset' parameter instead. [Default: None]
- **add_archive_leading_dir** (*bool, optional*) – place extracted content under a directory which would correspond to the archive name with all suffixes stripped. E.g. the content of *archive.tar.gz* will be extracted under *archive/*. [Default: False]
- **strip_leading_dirs** (*bool, optional*) – remove one or more leading directories from the archive layout on extraction. [Default: False]
- **leading_dirs_depth** – maximum depth of leading directories to strip. If not specified (None), no limit. [Default: None]
- **leading_dirs_consider** (*list of str or None, optional*) – regular expression(s) for directories to consider to strip away. [Default: None]
- **use_current_dir** (*bool, optional*) – extract the archive under the current directory, not the directory where the archive is located. This parameter is applied automatically if *key=True* was used. [Default: False]
- **delete** (*bool, optional*) – delete original archive from the filesystem/Git in current tree. Note that it will be of no effect if *key=True* is given. [Default: False]
- **key** (*bool, optional*) – signal if provided archive is not actually a filename on its own but an annex key. The archive will be extracted in the current directory. [Default: False]
- **exclude** (*list of str or None, optional*) – regular expressions for filenames which to exclude from being added to annex. Applied after –rename if that one is specified. For exact matching, use anchoring. [Default: None]
- **rename** (*list of str or None, optional*) – regular expressions to rename files before added them under to Git. The first defines how to split provided string into two parts: Python regular expression (with groups), and replacement string. [Default: None]
- **existing** – what operation to perform if a file from an archive tries to overwrite an existing file with the same name. 'fail' (default) leads to an error result, 'overwrite' silently replaces existing file, 'archive-suffix' instructs to add a suffix (prefixed with a '-') matching archive name from which file gets extracted, and if that one is present as well, 'numeric-suffix' is in effect in addition, when incremental numeric suffix (prefixed with a '.') is added until no name collision is longer detected. [Default: 'fail']
- **annex_options** (*str or None, optional*) – additional options to pass to git-annex. [Default: None]
- **copy** (*bool, optional*) – copy the content of the archive instead of moving. [Default: False]
- **commit** (*bool, optional*) – don't commit upon completion. [Default: True]
- **allow_dirty** (*bool, optional*) – flag that operating on a dirty repository (uncommitted or untracked content) is ok. [Default: False]
- **stats** – ActivityStats instance for global tracking. [Default: None]
- **drop_after** (*bool, optional*) – drop extracted files after adding to annex. [Default: False]
- **delete_after** (*bool, optional*) – extract under a temporary directory, git-annex add, and delete afterwards. To be used to "index" files within annex without actually creating

corresponding files under git. Note that *annex dropunused* would later remove that load. [Default: False]

- **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.api.add_readme

datalad.api.**add_readme**(*filename='README.md'*, *, *dataset=None*, *existing='skip'*)

Add basic information about DataLad datasets to a README file

The README file is added to the dataset and the addition is saved in the dataset. Note: Make sure that no unsaved modifications to your dataset's .gitattributes file exist.

Parameters

- **filename** (*str, optional*) – Path of the README file within the dataset. [Default: 'README.md']
- **dataset** (*Dataset or None, optional*) – Dataset to add information to. If no dataset is given, an attempt is made to identify the dataset based on the current working directory.

[Default: None]

- **existing** (*{'skip', 'append', 'replace'}, optional*) – How to react if a file with the target name already exists: 'skip': do nothing; 'append': append information to the existing file; 'replace': replace the existing file with new content. [Default: 'skip']
- **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.api.addurls

datalad.api.**addurls**(*urlfile*, *urlformat*, *filenameformat*, *, *dataset=None*, *input_type='ext'*, *exclude_autometa=None*, *meta=None*, *key=None*, *message=None*, *dry_run=False*, *fast=False*, *ifexists=None*, *missing_value=None*, *save=True*, *version_urls=False*, *cfg_proc=None*, *jobs=None*, *drop_after=False*, *on_collision='error'*)

Create and update a dataset from a list of URLs.

Format specification

Several arguments take format strings. These are similar to normal Python format strings where the names from *URL-FILE* (column names for a comma- or tab-separated file or properties for JSON) are available as

placeholders. If *URL-FILE* is a CSV or TSV file, a positional index can also be used (i.e., "{0}" for the first column). Note that a placeholder cannot contain a ':' or '!'.

In addition, the *FILENAME-FORMAT* arguments has a few special placeholders.

- _repindex

  The constructed file names must be unique across all fields rows. To avoid collisions, the special placeholder "_repindex" can be added to the formatter. Its value will start at 0 and increment every time a file name repeats.

- _url_hostname, _urlN, _url_basename*

  Various parts of the formatted URL are available. Take "http://datalad.org/asciicast/seamless_nested_repos.sh" as an example.

  "datalad.org" is stored as "_url_hostname". Components of the URL's path can be referenced as "_urlN". "_url0" and "_url1" would map to "asciicast" and "seamless_nested_repos.sh", respectively. The final part of the path is also available as "_url_basename".

  This name is broken down further. "_url_basename_root" and "_url_basename_ext" provide access to the root name and extension. These values are similar to the result of os.path.splitext, but, in the case of multiple periods, the extension is identified using the same length heuristic that git-annex uses. As a result, the extension of "file.tar.gz" would be ".tar.gz", not ".gz". In addition, the fields "_url_basename_root_py" and "_url_basename_ext_py" provide access to the result of os.path.splitext.

- _url_filename*

  These are similar to _url_basename* fields, but they are obtained with a server request. This is useful if the file name is set in the Content-Disposition header.

Examples

Consider a file "avatars.csv" that contains:

```
who,ext,link
neurodebian,png,https://avatars3.githubusercontent.com/u/260793
datalad,png,https://avatars1.githubusercontent.com/u/8927200
```

To download each link into a file name composed of the 'who' and 'ext' fields, we could run:

```
$ datalad addurls -d avatar_ds avatars.csv '{link}' '{who}.{ext}'
```

The *-d avatar_ds* is used to create a new dataset in "$PWD/avatar_ds".

If we were already in a dataset and wanted to create a new subdataset in an "avatars" subdirectory, we could use "//" in the *FILENAME-FORMAT* argument:

```
$ datalad addurls avatars.csv '{link}' 'avatars//{who}.{ext}'
```

If the information is represented as JSON lines instead of comma separated values or a JSON array, you can use a utility like jq to transform the JSON lines into an array that addurls accepts:

```
$ ... | jq --slurp . | datalad addurls - '{link}' '{who}.{ext}'
```

Note: For users familiar with 'git annex addurl': A large part of this plugin's functionality can be viewed as transforming data from *URL-FILE* into a "url filename" format that fed to 'git annex addurl –batch –with-files'.

Parameters

- **urlfile** – A file that contains URLs or information that can be used to construct URLs. Depending on the value of –input-type, this should be a comma- or tab-separated file (with a header as the first row) or a JSON file (structured as a list of objects with string values). If '-', read from standard input, taking the content as JSON when –input-type is at its default value of 'ext'. Alternatively, an iterable of dicts can be given.
- **urlformat** – A format string that specifies the URL for each entry. See the 'Format Specification' section above.
- **filenameformat** – Like *URL-FORMAT*, but this format string specifies the file to which the URL's content will be downloaded. The name should be a relative path and will be taken as relative to the top-level dataset, regardless of whether it is specified via *dataset* or inferred. The file name may contain directories. The separator "//" can be used to indicate that the left-side directory should be created as a new subdataset. See the 'Format Specification' section above.
- **dataset** (Dataset *or None, optional*) – Add the URLs to this dataset (or possibly subdatasets of this dataset). An empty or non-existent directory is passed to create a new dataset. New subdatasets can be specified with *FILENAME- FORMAT*. [Default: None]
- **input_type** (*{'ext', 'csv', 'tsv', 'json'}, optional*) – Whether *URL-FILE* should be considered a CSV file, TSV file, or JSON file. The default value, "ext", means to consider *URL-FILE* as a JSON file if it ends with ".json" or a TSV file if it ends with ".tsv". Otherwise, treat it as a CSV file. [Default: 'ext']
- **exclude_autometa** – By default, metadata field=value pairs are constructed with each column in *URL-FILE*, excluding any single column that is specified via *URL-FORMAT*. This argument can be used to exclude columns that match a regular expression. If set to '*' or an empty string, automatic metadata extraction is disabled completely. This argument does not affect metadata set explicitly with –meta. [Default: None]
- **meta** – A format string that specifies metadata. It should be structured as "<field>=<value>". As an example, "location={3}" would mean that the value for the "location" metadata field should be set the value of the fourth column. This option can be given multiple times. [Default: None]
- **key** – A format string that specifies an annex key for the file content. In this case, the file is not downloaded; instead the key is used to create the file without content. The value should be structured as "[et:]<input backend>[-s<bytes>]–<hash>". The optional "et:" prefix, which requires git-annex 8.20201116 or later, signals to toggle extension state of the input backend (i.e., MD5 vs MD5E). As an example, "et:MD5-s{size}–{md5sum}" would use the 'md5sum' and 'size' columns to construct the key, migrating the key from MD5 to MD5E, with an extension based on the file name. Note: If the *input* backend itself is an annex extension backend (i.e., a backend with a trailing "E"), the key's extension will not be updated to match the extension of the corresponding file name. Thus, unless the input keys and file names are generated from git-annex, it is recommended to avoid using extension backends as input. If an extension is desired, use the plain variant as input and prepend "et:" so that git-annex will migrate from the plain backend to the extension variant. [Default: None]
- **message** (*None or str, optional*) – Use this message when committing the URL additions. [Default: None]
- **dry_run** (*bool, optional*) – Report which URLs would be downloaded to which files and then exit. [Default: False]
- **fast** (*bool, optional*) – If True, add the URLs, but don't download their content. WARNING: ONLY USE THIS OPTION IF YOU UNDERSTAND THE CONSEQUENCES. If the content of the URLs is not downloaded, then datalad will refuse to retrieve

the contents with *datalad get <file>* by default because the content of the URLs is not verified. Add *annex.security.allow-unverified-downloads = ACKTHPPT* to your git config to bypass the safety check. Underneath, this passes the *–fast* flag to *git annex addurl*. [Default: False]

- **ifexists** (*{None, 'overwrite', 'skip'}, optional*) – What to do if a constructed file name already exists. The default behavior is to proceed with the *git annex addurl*, which will fail if the file size has changed. If set to 'overwrite', remove the old file before adding the new one. If set to 'skip', do not add the new file. [Default: None]
- **missing_value** (*None or str, optional*) – When an empty string is encountered, use this value instead. [Default: None]
- **save** (*bool, optional*) – by default all modifications to a dataset are immediately saved. Giving this option will disable this behavior. [Default: True]
- **version_urls** (*bool, optional*) – Try to add a version ID to the URL. This currently only has an effect on HTTP URLs for AWS S3 buckets. s3:// URL versioning is not yet supported, but any URL that already contains a "versionId=" parameter will be used as is. [Default: False]
- **cfg_proc** – Pass this cfg_proc value when calling *create* to make datasets. [Default: None]
- **jobs** (*int or None or {'auto'}, optional*) – how many parallel jobs (where possible) to use. "auto" corresponds to the number defined by 'datalad.runtime.max-annex-jobs' configuration item NOTE: This option can only parallelize input retrieval (get) and output recording (save). DataLad does NOT parallelize your scripts for you. [Default: None]
- **drop_after** (*bool, optional*) – drop files after adding to annex. [Default: False]
- **on_collision** (*{'error', 'error-if-different', 'take-first', 'take-last'}, optional*) – What to do when more than one row produces the same file name. By default an error is triggered. "error-if-different" suppresses that error if rows for a given file name collision have the same URL and metadata. "take-first" or "take-last" indicate to instead take the first row or last row from each set of colliding rows. [Default: 'error']
- **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.api.check_dates

datalad.api.**check_dates**(*paths*, *, *reference_date='@1514764800'*, *revs=None*, *annex='all'*, *no_tags=False*, *older=False*)

Find repository dates that are more recent than a reference date.

The main purpose of this tool is to find "leaked" real dates in repositories that are configured to use fake dates. It checks dates from three sources: (1) commit timestamps (author and committer dates), (2) timestamps within files of the "git-annex" branch, and (3) the timestamps of annotated tags.

Parameters

- **paths** (*sequence of str or None*) – Root directory in which to search for Git repositories. The current working directory will be used by default.
- **reference_date** (*str, optional*) – Compare dates to this date. If dateutil is installed, this value can be any format that its parser recognizes. Otherwise, it should be a unix timestamp that starts with a "@". The default value corresponds to 01 Jan, 2018 00:00:00 -0000. [Default: '@1514764800']
- **revs** – Search timestamps from commits that are reachable from these revisions. Any revision specification supported by git log, including flags like –all and –tags, can be used. [Default: None]
- **annex** (*{'all', 'tree', 'none'}, optional*) – Mode for "git-annex" branch search. If 'all', all blobs within the branch are searched. 'tree' limits the search to blobs that are referenced by the tree at the tip of the branch. 'none' disables search of "git-annex" blobs. [Default: 'all']
- **no_tags** (*bool, optional*) – Don't check the dates of annotated tags. [Default: False]
- **older** (*bool, optional*) – Find dates which are older than the reference date rather than newer. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']

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

datalad.api.configuration

datalad.api.**configuration**(*action='dump'*, *spec=None*, ***, *scope=None*, *dataset=None*, *recursive=False*, *recursion_limit=None*)

Get and set dataset, dataset-clone-local, or global configuration

This command works similar to git-config, but some features are not supported (e.g., modifying system configuration), while other features are not available in git-config (e.g., multi-configuration queries).

Query and modification of three distinct configuration scopes is supported:

- 'branch': the persistent configuration in .datalad/config of a dataset branch
- 'local': a dataset clone's Git repository configuration in .git/config
- 'global': non-dataset-specific configuration (usually in $USER/.gitconfig)

Modifications of the persistent 'branch' configuration will not be saved by this command, but have to be committed with a subsequent *save* call.

Rules of precedence regarding different configuration scopes are the same as in Git, with two exceptions: 1) environment variables can be used to override any datalad configuration, and have precedence over any other configuration scope (see below). 2) the 'branch' scope is considered in addition to the standard git configuration scopes. Its content has lower precedence than Git configuration scopes, but it is committed to a branch, hence can be used to ship (default and branch-specific) configuration with a dataset.

Besides storing configuration settings statically via this command or `git config`, DataLad also reads any DATALAD_* environment on process startup or import, and maps it to a configuration item. Their values take precedence over any other specification. In variable names _ encodes a `.` in the configuration name, and __ encodes a `-`, such that `DATALAD_SOME__VAR` is mapped to `datalad.some-var`. Additionally, a DATALAD_CONFIG_OVERRIDES_JSON environment variable is queried, which may contain configuration key-value mappings as a JSON-formatted string of a JSON-object:

```
DATALAD_CONFIG_OVERRIDES_JSON='{"datalad.credential.example_com.user": "jane", ...}'
```

This is useful when characters are part of the configuration key that cannot be encoded into an environment variable name. If both individual configuration variables *and* JSON-overrides are used, the former take precedent over the latter, overriding the respective *individual* settings from configurations declared in the JSON-overrides.

This command supports recursive operation for querying and modifying configuration across a hierarchy of datasets.

Examples

Dump the effective configuration, including an annotation for common items:

```
> configuration()
```

Query two configuration items:

```
> configuration('get', ['user.name', 'user.email'])
```

Recursively set configuration in all (sub)dataset repositories:

```
> configuration('set', [('my.config.name', 'value')], recursive=True)
```

Modify the persistent branch configuration (changes are not committed):

```
> configuration('set', [('my.config.name', 'value')], scope='branch')
```

Parameters

- **action** (*{'dump', 'get', 'set', 'unset'}, optional*) – which action to perform. [Default: 'dump']
- **spec** – configuration name (for actions 'get' and 'unset'), or name/value pair (for action 'set'). [Default: None]
- **scope** (*{'global', 'local', 'branch', None}, optional*) – scope for getting or setting configuration. If no scope is declared for a query, all configuration sources (including overrides via environment variables) are considered according to the normal rules of precedence. For action 'get' only 'branch' and 'local' (which include 'global' here) are supported. For action 'dump', a scope selection is ignored and all available scopes are considered. [Default: None]
- **dataset** (*Dataset or None, optional*) – specify the dataset to query or to configure. [Default: None]
- **recursive** (*bool, optional*) – if set, recurse into potential subdatasets. [Default: False]
- **recursion_limit** (*int or None, optional*) – limit recursion into subdatasets to the given number of levels. [Default: None]

- **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.api.export_archive

datalad.api.**export_archive**(*filename=None*, *, *dataset=None*, *archivetype='tar'*, *compression='gz'*, *missing_content='error'*)

Export the content of a dataset as a TAR/ZIP archive.

Parameters

- **filename** (*str or None, optional*) – File name of the generated TAR archive. If no file name is given the archive will be generated in the current directory and will be named: datalad_<dataset_uuid>.(tar.*|zip). To generate that file in a different directory, provide an existing directory as the file name. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **archivetype** (*{'tar', 'zip'}, optional*) – Type of archive to generate. [Default: 'tar']

- **compression** (*{'gz', 'bz2', ''}, optional*) – Compression method to use. 'bz2' is not supported for ZIP archives. No compression is used when an empty string is given. [Default: 'gz']
- **missing_content** (*{'error', 'continue', 'ignore'}, optional*) – By default, any discovered file with missing content will result in an error and the export is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. [Default: 'error']
- **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.api.export_archive_ora

datalad.api.**export_archive_ora**(*target*, *opts=None*, *, *dataset=None*, *remote=None*, *annex_wanted=None*, *froms=None*, *missing_content='error'*)

Export an archive of a local annex object store for the ORA remote.

Keys in the local annex object store are reorganized in a temporary directory (using links to avoid storage duplication) to use the 'hashdirlower' setup used by git-annex for bare repositories and the directory-type special remote. This alternative object store is then moved into a 7zip archive that is suitable for use in a ORA remote dataset store. Placing such an archive into:

```
<dataset location>/archives/archive.7z
```

Enables the ORA special remote to locate and retrieve all keys contained in the archive.

Parameters

- **target** (*str or None*) – if an existing directory, an 'archive.7z' is placed into it, otherwise this is the path to the target archive.
- **opts** – list of options for 7z to replace the default '-mx0' to generate an uncompressed archive. [Default: None]
- **dataset** (*Dataset or None, optional*) – 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]
- **remote** (*str or None, optional*) – name of the target sibling, wanted/preferred settings will be used to filter the files added to the archives. [Default: None]
- **annex_wanted** – git-annex-preferred-content expression for git-annex find to filter files. Should start with 'or' or 'and' when used in combination with *–for*. [Default: None]
- **froms** – one or multiple tree-ish from which to select files. [Default: None]
- **missing_content** (*{'error', 'continue', 'ignore'}, optional*) – By default, any discovered file with missing content will result in an error and the export is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. [Default: 'error']
- **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.api.export_to_figshare

datalad.api.**export_to_figshare**(*filename=None*, *, *dataset=None*, *missing_content='error'*, *no_annex=False*, *article_id=None*)

Export the content of a dataset as a ZIP archive to figshare

Very quick and dirty approach. Ideally figshare should be supported as a proper git annex special remote. Unfortunately, figshare does not support having directories, and can store only a flat list of files. That makes it impossible for any sensible publishing of complete datasets.

The only workaround is to publish dataset as a zip-ball, where the entire content is wrapped into a .zip archive for which figshare would provide a navigator.

Parameters

- **filename** (*str or None, optional*) – File name of the generated ZIP archive. If no file name is given the archive will be generated in the top directory of the dataset and will be named: datalad_<dataset_uuid>.zip. [Default: None]
- **dataset** (*Dataset or None, optional*) – "specify the dataset to export. If no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **missing_content** (*{'error', 'continue', 'ignore'}, optional*) – By default, any discovered file with missing content will result in an error and the plugin is aborted. Setting this to 'continue' will issue warnings instead of failing on error. The value 'ignore' will only inform about problem at the 'debug' log level. The latter two can be helpful when generating a TAR archive from a dataset where some file content is not available locally. [Default: 'error']
- **no_annex** (*bool, optional*) – By default the generated .zip file would be added to annex, and all files would get registered in git-annex to be available from such a tarball. Also upon upload we will register for that archive to be a possible source for it in annex. Setting this flag disables this behavior. [Default: False]
- **article_id** (*int or None, optional*) – Which article to publish to. [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.api.no_annex

datalad.api.**no_annex**(*dataset*, *pattern*, *ref_dir='.'*, *makedirs=False*)

Configure a dataset to never put some content into the dataset's annex

This can be useful in mixed datasets that also contain textual data, such as source code, which can be efficiently and more conveniently managed directly in Git.

Patterns generally look like this:

```
code/*
```

which would match all file in the code directory. In order to match all files under `code/`, including all its subdirectories use such a pattern:

```
code/**
```

Note that this command works incrementally, hence any existing configuration (e.g. from a previous plugin run) is amended, not replaced.

Parameters

- **dataset** (*Dataset or None*) – "specify the dataset to configure. If no dataset is given, an attempt is made to identify the dataset based on the current working directory.
- **pattern** – list of path patterns. Any content whose path is matching any pattern will not be annexed when added to a dataset, but instead will be tracked directly in Git. Path pattern have to be relative to the directory given by the *ref_dir* option. By default, patterns should be relative to the root of the dataset.
- **ref_dir** – Relative path (within the dataset) to the directory that is to be configured. All patterns are interpreted relative to this path, and configuration is written to a `.gitattributes` file in this directory. [Default: '.']
- **makedirs** (*bool, optional*) – If set, any missing directories will be created in order to be able to place a file into `--ref-dir`. [Default: False]
- **on_failure** (*{'ignore', 'continue', 'stop'}, optional*) – behavior to perform on failure: 'ignore' any failure is reported, but does not cause an exception; 'continue' if any failure occurs an exception will be raised at the end, but processing other actions will continue for as long as possible; 'stop': processing will stop on first failure and an exception is raised. A failure is any result with status 'impossible' or 'error'. Raised exception is an IncompleteResultsError that carries the result dictionaries of the failures in its *failed* attribute. [Default: 'continue']
- **result_filter** (*callable or None, optional*) – if given, each to-be-returned status dictionary is passed to this callable, and is only returned if the callable's return value does not evaluate to False or a ValueError exception is raised. If the given callable supports ***kwargs* it will additionally be passed the keyword arguments of the original API call. [Default: None]
- **result_renderer** – select rendering mode command results. 'tailored' enables a command- specific rendering style that is typically tailored to human consumption, if there is one for a specific command, or otherwise falls back on the the 'generic' result renderer; 'generic' renders each result in one line with key info like action, status, path, and an optional message); 'json' a complete JSON line serialization of the full result record; 'json_pp' like 'json', but pretty-printed spanning multiple lines; 'disabled' turns off result rendering entirely; '<template>' reports any value(s) of any result properties in any format indicated by the template (e.g. '{path}', compare with JSON output for all key-value choices). The template syntax follows the Python "format() language". It is possible to report individual dictionary values, e.g. '{metadata[name]}'. If a 2nd-level key contains a colon, e.g. 'music:Genre', ':' must be substituted by '#' in the template, like so: '{metadata[music#Genre]}'. [Default: 'tailored']
- **result_xfm** (*{'datasets', 'successdatasets-or-none', 'paths', 'relpaths', 'metadata'} or callable or None, optional*) – if given, each to-be-returned result status dictionary is passed to this callable, and its return value becomes the result instead. This is different from *result_filter*, as it can perform arbitrary transformation of the result value. This is mostly useful for top- level command invocations that need to provide the results in a particular format. Instead of a callable, a label for a pre-crafted result transformation can be given. [Default: None]
- **return_type** (*{'generator', 'list', 'item-or-list'}, optional*) – return value behavior switch. If 'item-or-list' a single value is returned instead of a one-item return value list, or a list in case of multiple return values. *None* is return in case of an empty list. [Default: 'list']

datalad.api.shell_completion

datalad.api.**shell_completion**()

Display shell script for enabling shell completion for DataLad.

Output of this command should be "sourced" by the bash or zsh to enable shell completions provided by argcomplete.

Example

$ source <(datalad shell-completion) $ datalad –<PRESS TAB to display available option>

Parameters

- **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.api.wtf

datalad.api.**wtf**(*, *dataset=None*, *sensitive=None*, *sections=None*, *flavor='full'*, *decor=None*, *clipboard=None*)

Generate a report about the DataLad installation and configuration

IMPORTANT: Sharing this report with untrusted parties (e.g. on the web) should be done with care, as it may include identifying information, and/or credentials or access tokens.

Parameters

- **dataset** (*Dataset or None, optional*) – "specify the dataset to report on. no dataset is given, an attempt is made to identify the dataset based on the current working directory. [Default: None]
- **sensitive** (*{None, 'some', 'all'}, optional*) – if set to 'some' or 'all', it will display sections such as config and metadata which could potentially contain sensitive information (credentials, names, etc.). If 'some', the fields which are known to be sensitive will still be masked out. [Default: None]
- **sections** (*list of {None, 'configuration', 'credentials', 'datalad', 'dataset', 'dependencies', 'environment', 'extensions', 'git-annex', 'location', 'metadata', 'metadata.extractors', 'metadata.filters', 'metadata.indexers', 'python', 'system', '*'}, optional*) – section to include. If not set - depends on flavor. '*' could be used to force all sections. If there are subsections like section.subsection available, then specifying just 'section' would select all subsections for that section. [Default: None]
- **flavor** (*{'full', 'short'}, optional*) – Flavor of WTF. 'full' would produce markdown with exhaustive list of sections. 'short' will provide a condensed summary only of datalad and dependencies by default. Use *section* to list other sections. [Default: 'full']
- **decor** (*{'html_details', None}, optional*) – decoration around the rendering to facilitate embedding into issues etc, e.g. use 'html_details' for posting collapsible entry to GitHub issues. [Default: None]
- **clipboard** (*bool, optional*) – if set, do not print but copy to clipboard (requires pyperclip module). [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']

Support functionality

cmd	Class the starts a subprocess and keeps it around to communicate with it via stdin.
consts	constants for datalad
log	Logging setup and utilities, including progress reporting
utils	
version	
support.gitrepo	Internal low-level interface to Git repositories
support.annexrepo	Interface to git-annex by Joey Hess.
support.archives	Various handlers/functionality for different types of files (e.g. for archives).
support.extensions	Support functionality for extension development
customremotes.base	Base classes to custom git-annex remotes (e.g. extraction from archives).
customremotes.archives	Custom remote to get the load from archives present under annex
runner.nonasyncrunner	Thread based subprocess execution with stdout and stderr passed to protocol objects
runner.protocol	Base class of a protocol to be used with the DataLad runner

datalad.cmd

Class the starts a subprocess and keeps it around to communicate with it via stdin. For each instruction send over stdin, a response is read and returned. The response structure is determined by "output_proc"

class datalad.cmd.**BatchedCommand**(*cmd*, *path=None*, *output_proc=None*, *timeout=None*, *exception_on_timeout=False*)

Bases: *SafeDelCloseMixin*

Container for a running subprocess. Supports communication with the subprocess via stdin and stdout.

Parameters

- **cmd** (Union[str, Tuple, List]) –
- **path** (Optional[str]) –
- **output_proc** (Optional[Callable]) –
- **timeout** (Optional[float]) –
- **exception_on_timeout** (bool) –

classmethod clean_inactive()

close(*return_stderr=False*)

Close communication and wait for process to terminate. If the "timeout" parameter to the constructor was not None, and if the configuration setting "datalad.runtime.stalled-external" is set to "abandon", the method will return latest after "timeout" seconds. If the subprocess did not exit within this time, the attribute "wait_timed_out" will be set to "True".

Parameters
return_stderr (*bool*) – if set to "True", the call will return all collected stderr content as string. In addition, if return_stderr is True and the log level is 5 or lower, and the configuration setting "datalad.log.outputs" evaluates to "True", the content of stderr will be logged.

Returns
stderr output if return_stderr is True, None otherwise

Return type
str, optional

get_one_line()

Get a single stdout line from the generator.

If timeout was specified, and exception_on_timeout is False, and if a timeout occurs, return None. Otherwise, return the string that was read from the generator.

Return type
Optional[str]

get_requested_error_output(*return_stderr*)

Parameters
return_stderr (bool) –

get_timeout_exception(*fd*)

Get a process timeout exception if timeout exceptions should be generated for a process that continues longer than timeout seconds after self.close() was initiated.

Parameters
fd (Optional[int]) –

Return type
Optional[TimeoutExpired]

proc1(*single_command*)

Simulate the old interface. This method is used only once in AnnexRepo.get_metadata()

Parameters
single_command (str) –

process_request(*request*)

Parameters
request (Union[Tuple, str]) –

Return type
Any | None

process_running()

Return type
bool

exception datalad.cmd.**BatchedCommandError**(*cmd=''*, *last_processed_request=''*, *msg=''*, *code=None*, *stdout=''*, *stderr=''*, *cwd=None*, ***kwargs*)

Bases: CommandError

class datalad.cmd.**BatchedCommandProtocol**(*batched_command*, *done_future=None*, *encoding=None*, *output_proc=None*)

Bases: *GeneratorMixIn*, StdOutErrCapture

Parameters

- **batched_command** (*BatchedCommand*) –
- **done_future** (Optional[Any]) –
- **encoding** (Optional[str]) –
- **output_proc** (Optional[Callable]) –

pipe_connection_lost(*fd*, *exc*)

Called when a file descriptor associated with the child process is closed.

fd is the int file descriptor that was closed.

Parameters

- **fd** (int) –
- **exc** (Optional[BaseException]) –

pipe_data_received(*fd*, *data*)

Parameters

- **fd** (int) –
- **data** (bytes) –

timeout(*fd*)

Called if the timeout parameter to WitlessRunner.run() is not *None* and a process file descriptor could not be read (stdout or stderr) or not be written (stdin) within the specified time in seconds, or if waiting for a subprocess to exit takes longer than the specified time.

stdin timeouts are only caught when the type of the *stdin-* parameter to WitlessRunner.run() is either a *Queue*, a *str*, or *bytes*. *Stdout* or *stderr* timeouts are only caught of proc_out and proc_err are *True* in the protocol class. Process wait timeouts are always caught if *timeout* is not *None*. In this case the *fd*-argument will be *None*.

fd:
The file descriptor that timed out or *None* if no progress was made at all, i.e. no stdin element was enqueued and no output was read from either stdout or stderr.

Return type
`bool`

Returns
If the callback returns *True*, the file descriptor (if any was given) will be closed and no longer monitored. If the return values is anything else than *True*, the file-descriptor will be monitored further and additional timeouts might occur indefinitely. If *None* was given, i.e. a process runtime-timeout was detected, and *True* is returned, the process will be terminated.

Parameters
fd (`Optional[int]`) –

class datalad.cmd.**ReadlineEmulator**(*batched_command*)

Bases: `object`

This class implements readline() on the basis of an instance of BatchedCommand. Its purpose is to emulate stdout's for output_procs, This allows us to provide a BatchedCommand API that is identical to the old version, but with an implementation that is based on the threaded runner.

Parameters
batched_command (*BatchedCommand*) –

readline()

Read from the stdout provider until we have a line or None (which indicates some error).

class datalad.cmd.**SafeDelCloseMixin**

Bases: `object`

A helper class to use where __del__ would call .close() which might fail if "too late in GC game"

datalad.cmd.**readline_rstripped**(*stdout*)

datalad.consts

constants for datalad

datalad.log

Logging setup and utilities, including progress reporting

class datalad.log.**ColorFormatter**(*use_color=None*, *log_name=False*, *log_pid=False*)

Bases: `Formatter`

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class datalad.log.**LoggerHelper**(*name='datalad'*, *logtarget=None*)

Bases: `object`

Helper to establish and control a Logger

get_initialized_logger(*logtarget=None*)

Initialize and return the logger

Parameters

logtarget (*{'stderr', str }, optional*) – Where to direct the logs. 'stderr' stands for the standard stream. Any other string is considered a filename. Multiple entries could be specified comma-separated

Return type

logging.Logger

set_level(*level=None*, *default='INFO'*)

Helper to set loglevel for an arbitrary logger

By default operates for 'datalad'. TODO: deduce name from upper module name so it could be reused without changes

datalad.log.**filter_noninteractive_progress**(*logger*, *record*)

Companion of log_progress() to suppress undesired progress logging

This filter is to be used with a log handler's addFilter() method for the case of a non-interactive session (e.g., pipe to log file).

It inspects the log record for *dlm_progress_noninteractive_level* keys that can be injected via log_progress(noninteractive_level=).

If a log-level was declared in this fashion, it will be evaluated against the logger's effective level, and records are discarded if their level is too low. If no log-level was declared, a log record passes this filter unconditionally.

Parameters

- **logger** (*logging.Logger*) – The logger instance whose effective level to check against.
- **record** – The log record to inspect.

Return type

bool

datalad.log.**log_progress**(*lgrcall*, *pid*, **args*, ***kwargs*)

Emit progress log messages

This helper can be used to handle progress reporting without having to maintain display mode specific code.

Typical progress reporting via this function involves three types of calls:

1. Start reporting progress about a process
2. Update progress information about a process
3. Report completion of a process

In order to be able to associate all three steps with a particular process, the *pid* identifier is used. This is an arbitrary string that must be chosen to be unique across all different, but simultaneously running progress reporting activities within a Python session. For many practical purposes this can be achieved by, for example, including path information in the identifier.

To initialize a progress report this function is called without an *update* parameter. To report a progress update, this function is called with an *update* parameter. To finish a reporting on a particular activity a final call without an *update* parameter is required.

Parameters

- **lgrcall** (*callable*) – Something like lgr.debug or lgr.info

- **pid** (*str*) – Some kind of ID for the process the progress is reported on.
- ***args** (*str*) – Log message, and potential arguments
- **total** (*int*) – Max progress quantity of the process.
- **label** (*str*) – Process description. Should be very brief, goes in front of progress bar on the same line.
- **unit** (*str*) – Progress report unit. Should be very brief, goes after the progress bar on the same line.
- **update** (*int*) – To (or by) which quantity to advance the progress. Also see *increment*.
- **increment** (*bool*) – If set, *update* is interpreted as an incremental value, not absolute.
- **initial** (*int*) – If set, start value for progress bar
- **noninteractive_level** (*int, optional*) – In a non-interactive session where progress bars are not displayed, only log a progress report, if a logger's effective level includes the specified level. This can be useful logging all progress is inappropriate or too noisy for a log.
- **maint** (*{'clear', 'refresh'}*) – This is a special attribute that can be used by callers that are not actually reporting progress, but need to ensure that their (console) output does not interfere with any possibly ongoing progress reporting. Setting this attribute to 'clear' will cause the central ProgressHandler to temporarily stop the display of any active progress bars. With 'refresh', all active progress bars will be redisplayed. After a 'clear' individual progress bars would be reactivated upon the next update log message, even without an explicit 'refresh'.

datalad.log.**with_progress**(*items*, *lgrcall=None*, *label='Total'*, *unit=' Files'*)

Wrap a progress bar, with status counts, around an iterable.

Parameters

- **items** (*some iterable*) –
- **lgrcall** (*callable*) – Callable for logging. If not specified - lgr.info is used
- **label** (*str*) – Passed to log.log_progress.
- **unit** (*str*) – Passed to log.log_progress.

Yields

Items of it while displaying the progress

datalad.log.**with_result_progress**(*fn*, *label='Total'*, *unit=' Files'*, *log_filter=None*)

Wrap a progress bar, with status counts, around a function.

Parameters

- **fn** (*generator function*) – This function should accept a collection of items as a positional argument and any number of keyword arguments. After processing each item in the collection, it should yield a status dict.
- **log_filter** (*callable, optional*) – If defined, only result records for which callable evaluates to True will be passed to log_progress
- **label** (*str*) – Passed to log.log_progress.
- **unit** (*str*) – Passed to log.log_progress.

Returns

- A variant of *fn* that shows a progress bar. Note that the wrapped

- *function is not a generator function; the status dicts will be*
- *returned as a list.*

datalad.utils

class datalad.utils.**ArgSpecFake**(*args*, *varargs*, *keywords*, *defaults*)

Bases: NamedTuple

args: list[str]

Alias for field number 0

defaults: Optional[tuple[Any, ...]]

Alias for field number 3

keywords: Optional[str]

Alias for field number 2

varargs: Optional[str]

Alias for field number 1

class datalad.utils.**File**(*name*, *executable=False*)

Bases: object

Helper for a file entry in the create_tree/@with_tree

It allows to define additional settings for entries

Parameters

- **name** (str) –
- **executable** (bool) –

class datalad.utils.**SequenceFormatter**(*separator=' '*, *element_formatter=<string.Formatter object>*, **args*, ***kwargs*)

Bases: Formatter

string.Formatter subclass with special behavior for sequences.

This class delegates formatting of individual elements to another formatter object. Non-list objects are formatted by calling the delegate formatter's "format_field" method. List-like objects (list, tuple, set, frozenset) are formatted by formatting each element of the list according to the specified format spec using the delegate formatter and then joining the resulting strings with a separator (space by default).

Parameters

- **separator** (str) –
- **element_formatter** (Formatter) –
- **args** (Any) –
- **kwargs** (Any) –

format_element(*elem*, *format_spec*)

Format a single element

For sequences, this is called once for each element in a sequence. For anything else, it is called on the entire object. It is intended to be overridden in subclases.

Parameters

- **elem** (Any) –
- **format_spec** (str) –

Return type
Any

format_field(*value*, *format_spec*)

Parameters

- **value** (Any) –
- **format_spec** (str) –

Return type
Any

class datalad.utils.**SwallowLogsAdapter**(*file_*)

Bases: object

Little adapter to help getting out values

And to stay consistent with how swallow_outputs behaves

Parameters
file_ (str | Path | None) –

assert_logged(*msg=None*, *level=None*, *regex=True*, ***kwargs*)

Provide assertion on whether a msg was logged at a given level

If neither *msg* nor *level* provided, checks if anything was logged at all.

Parameters

- **msg** (*str, optional*) – Message (as a regular expression, if *regex*) to be searched. If no msg provided, checks if anything was logged at a given level.
- **level** (*str, optional*) – String representing the level to be logged
- **regex** (*bool, optional*) – If False, regular *assert_in* is used
- ****kwargs** (*str, optional*) – Passed to *assert_re_in* or *assert_in*

Return type
None

cleanup()

Return type
None

property handle: IO[str]

property lines: list[str]

property out: str

class datalad.utils.**SwallowOutputsAdapter**

Bases: object

Little adapter to help getting out/err values

cleanup()

Return type
None

property err: str

property handles: tuple[TextIO, TextIO]

property out: str

datalad.utils.**any_re_search**(*regexes*, *value*)

Return if any of regexes (list or str) searches successfully for value

Parameters

- **regexes** (str | list[str]) –
- **value** (str) –

Return type
bool

datalad.utils.**assert_no_open_files**(*path*)

Parameters
path (str | Path) –

Return type
None

datalad.utils.**assure_bool**(*s*)

Note: This function is deprecated. Use ensure_bool instead.

Parameters
s (Any) –

Return type
bool

datalad.utils.**assure_bytes**(*s*, *encoding='utf-8'*)

Note: This function is deprecated. Use ensure_bytes instead.

Parameters

- **s** (str | bytes) –
- **encoding** (str) –

Return type
bytes

datalad.utils.**assure_dict_from_str**(*s*, *sep='\\n'*)

Note: This function is deprecated. Use ensure_dict_from_str instead.

Parameters

- **s** (str | dict[K, V]) –
- **sep** (str) –

Return type
Optional[dict[str, str]] | Optional[dict[K, V]]

datalad.utils.**assure_dir**(**args*)

Note: This function is deprecated. Use ensure_dir instead.

Parameters
args (str) –

Return type
str

datalad.utils.**assure_iter**(*s*, *cls*, *copy=False*, *iterate=True*)

Note: This function is deprecated. Use ensure_iter instead.

Parameters

- **s** (Any) –
- **cls** (type[TypeVar(ListOrSet, list, set)]) –
- **copy** (bool) –
- **iterate** (bool) –

Return type
TypeVar(ListOrSet, list, set)

datalad.utils.**assure_list**(*s*, *copy=False*, *iterate=True*)

Note: This function is deprecated. Use ensure_list instead.

Parameters

- **s** (Any) –
- **copy** (bool) –
- **iterate** (bool) –

Return type
list

datalad.utils.**assure_list_from_str**(*s*, *sep=\\n'*)

Note: This function is deprecated. Use ensure_list_from_str instead.

Parameters

- **s** (str | list[T]) –
- **sep** (str) –

Return type
Optional[list[str]] | Optional[list[T]]

datalad.utils.**assure_tuple_or_list**(*obj*)

Note: This function is deprecated. Use ensure_tuple_or_list instead.

Parameters
obj (Any) –

Return type
list | tuple

datalad.utils.**assure_unicode**(*s*, *encoding=None*, *confidence=None*)

Note: This function is deprecated. Use ensure_unicode instead.

Parameters

- **s** (str | bytes) –

- **encoding** (Optional[str]) –
- **confidence** (Optional[float]) –

Return type
str

datalad.utils.**auto_repr**(*cls*, *short=True*)

Decorator for a class to assign it an automagic quick and dirty __repr__

It uses public class attributes to prepare repr of a class

Original idea: http://stackoverflow.com/a/27799004/1265472

Parameters

- **cls** (type[TypeVar(T)]) –
- **short** (bool) –

Return type
type[TypeVar(T)]

datalad.utils.**bytes2human**(*n*, *format='%(value).1f %(symbol)sB'*)

Convert n bytes into a human readable string based on format. symbols can be either "customary", "customary_ext", "iec" or "iec_ext", see: http://goo.gl/kTQMs

```
>>> from datalad.utils import bytes2human
>>> bytes2human(1)
'1.0 B'
>>> bytes2human(1024)
'1.0 KB'
>>> bytes2human(1048576)
'1.0 MB'
>>> bytes2human(1099511627776127398123789121)
'909.5 YB'
```

```
>>> bytes2human(10000, "%(value).1f %(symbol)s/sec")
'9.8 K/sec'
```

```
>>> # precision can be adjusted by playing with %f operator
>>> bytes2human(10000, format="%(value).5f %(symbol)s")
'9.76562 K'
```

Taken from: http://goo.gl/kTQMs and subsequently simplified Original Author: Giampaolo Rodola' <g.rodola [AT] gmail [DOT] com> License: MIT

Parameters

- **n** (int | float) –
- **format** (str) –

Return type
str

datalad.utils.**check_symlink_capability**(*path*, *target*)

helper similar to datalad.tests.utils_pytest.has_symlink_capability

However, for use in a datalad command context, we shouldn't assume to be able to write to tmpfile and also not import a whole lot from datalad's test machinery. Finally, we want to know, whether we can create a symlink at a

specific location, not just somewhere. Therefore use arbitrary path to test-build a symlink and delete afterwards. Suitable location can therefore be determined by high lever code.

Parameters

- **path** (*Path*) –
- **target** (*Path*) –

Return type
bool

class datalad.utils.**chpwd**(*path*, *mkdir=False*, *logsuffix=''*)

Bases: object

Wrapper around os.chdir which also adjusts environ['PWD']

The reason is that otherwise PWD is simply inherited from the shell and we have no ability to assess directory path without dereferencing symlinks.

If used as a context manager it allows to temporarily change directory to the given path

Parameters

- **path** (str | Path | None) –
- **mkdir** (bool) –
- **logsuffix** (str) –

datalad.utils.**collect_method_callstats**(*func*)

Figure out methods which call the method repeatedly on the same instance

Use case(s):

- .repo is expensive since does all kinds of checks.
- .config is expensive transitively since it calls .repo each time

---

Todo:

- fancy one could look through the stack for the same id(self) to see if that location is already in memo. That would hint to the cases where object is not passed into underlying functions, causing them to redo the same work over and over again
- ATM might flood with all "1 lines" calls which are not that informative. The underlying possibly suboptimal use might be coming from their callers. It might or not relate to the previous TODO

Parameters
func (Callable[[ParamSpec(P)], TypeVar(T)]) –

Return type
Callable[[ParamSpec(P)], TypeVar(T)]

datalad.utils.**create_tree**(*path*, *tree*, *archives_leading_dir=True*, *remove_existing=False*)

Given a list of tuples (name, load) create such a tree

if load is a tuple itself – that would create either a subtree or an archive with that content and place it into the tree if name ends with .tar.gz

Parameters

- **path** (str) –

- **tree** (Union[Tuple[Tuple[Union[str, *File*], Union[str, bytes, TreeSpec]], ...], List[Tuple[Union[str, *File*], Union[str, bytes, TreeSpec]]], Dict[Union[str, *File*], Union[str, bytes, TreeSpec]]]) –
- **archives_leading_dir** (bool) –
- **remove_existing** (bool) –

Return type
None

datalad.utils.**create_tree_archive**(*path*, *name*, *load*, *overwrite=False*, *archives_leading_dir=True*)

Given an archive *name*, create under *path* with specified *load* tree

Parameters

- **path** (str) –
- **name** (str) –
- **load** (Union[Tuple[Tuple[Union[str, *File*], Union[str, bytes, TreeSpec]], ...], List[Tuple[Union[str, *File*], Union[str, bytes, TreeSpec]]], Dict[Union[str, *File*], Union[str, bytes, TreeSpec]]]) –
- **overwrite** (bool) –
- **archives_leading_dir** (bool) –

Return type
None

datalad.utils.**decode_input**(*s*)

Given input string/bytes, decode according to stdin codepage (or UTF-8) if not defined

If fails – issue warning and decode allowing for errors being replaced

Parameters
s (str | bytes) –

Return type
str

datalad.utils.**disable_logger**(*logger=None*)

context manager to temporarily disable logging

This is to provide one of swallow_logs' purposes without unnecessarily creating temp files (see gh-1865)

Parameters
logger (*Logger*) – Logger whose handlers will be ordered to not log anything. Default: datalad's topmost Logger ('datalad')

Return type
Iterator[Logger]

datalad.utils.**dlabspath**(*path*, *norm=False*)

Symlinks-in-the-cwd aware abspath

os.path.abspath relies on os.getcwd() which would not know about symlinks in the path

TODO: we might want to norm=True by default to match behavior of os .path.abspath?

Parameters

- **path** (str | Path) –

- **norm** (bool) –

Return type
str

datalad.utils.**encode_filename**(*filename*)

Encode unicode filename

Parameters
filename (str | bytes) –

Return type
bytes

datalad.utils.**ensure_bool**(*s*)

Convert value into boolean following convention for strings

to recognize on,True,yes as True, off,False,no as False

Parameters
s (Any) –

Return type
bool

datalad.utils.**ensure_bytes**(*s*, *encoding='utf-8'*)

Convert/encode unicode string to bytes.

If *s* isn't a string, return it as is.

Parameters

- **encoding** (*str, optional*) – Encoding to use. "utf-8" is the default
- **s** (str | bytes) –

Return type
bytes

datalad.utils.**ensure_dict_from_str**(*s*, *sep='\\n'*)

Given a multiline string with key=value items convert it to a dictionary

Parameters

- **s** (*str or dict*) –
- **empty** (*Returns None if input s is*) –
- **sep** (str) –

Return type
Optional[dict[str, str]] | Optional[dict[K, V]]

datalad.utils.**ensure_dir**(**args*)

Make sure directory exists.

Joins the list of arguments to an os-specific path to the desired directory and creates it, if it not exists yet.

Parameters
args (str) –

Return type
str

datalad.utils.**ensure_iter**(*s*, *cls*, *copy=False*, *iterate=True*)

Given not a list, would place it into a list. If None - empty list is returned

Parameters

- **s** (*list or anything*) –
- **cls** (*class*) – Which iterable class to ensure
- **copy** (*bool, optional*) – If correct iterable is passed, it would generate its shallow copy
- **iterate** (*bool, optional*) – If it is not a list, but something iterable (but not a str) iterate over it.

Return type
TypeVar(ListOrSet, list, set)

datalad.utils.**ensure_list**(*s*, *copy=False*, *iterate=True*)

Given not a list, would place it into a list. If None - empty list is returned

Parameters

- **s** (*list or anything*) –
- **copy** (*bool, optional*) – If list is passed, it would generate a shallow copy of the list
- **iterate** (*bool, optional*) – If it is not a list, but something iterable (but not a str) iterate over it.

Return type
list

datalad.utils.**ensure_list_from_str**(*s*, *sep=\\n'*)

Given a multiline string convert it to a list of return None if empty

Parameters

- **s** (*str or list*) –
- **sep** (str) –

Return type
Optional[list[str]] | Optional[list[T]]

datalad.utils.**ensure_result_list**(*r*)

Return a list of result records

Largely same as ensure_list, but special casing a single dict being passed in, which a plain *ensure_list* would iterate over. Hence, this deals with the three ways datalad commands return results: - single dict - list of dicts - generator

Used for result assertion helpers.

Parameters
r (Any) –

Return type
list

datalad.utils.**ensure_tuple_or_list**(*obj*)

Given an object, wrap into a tuple if not list or tuple

Parameters
obj (Any) –

Return type
list | tuple

datalad.utils.**ensure_unicode**(*s*, *encoding=None*, *confidence=None*)

Convert/decode bytestring to unicode.

If *s* isn't a bytestring, return it as is.

Parameters

- **encoding** (*str, optional*) – Encoding to use. If None, "utf-8" is tried, and then if not a valid UTF-8, encoding will be guessed
- **confidence** (*float, optional*) – A value between 0 and 1, so if guessing of encoding is of lower than specified confidence, ValueError is raised
- **s** (str | bytes) –

Return type
str

datalad.utils.**ensure_write_permission**(*path*)

Context manager to get write permission on *path* and restore original mode afterwards.

Parameters
path (*Path*) – path to the target file

Raises
PermissionError – if write permission could not be obtained

Return type
Iterator[None]

datalad.utils.**escape_filename**(*filename*)

Surround filename in "" and escape " in the filename

Parameters
filename (str) –

Return type
str

datalad.utils.**expandpath**(*path*, *force_absolute=True*)

Expand all variables and user handles in a path.

By default return an absolute path

Parameters

- **path** (str | Path) –
- **force_absolute** (bool) –

Return type
str

datalad.utils.**file_basename**(*name*, *return_ext=False*)

Strips up to 2 extensions of length up to 4 characters and starting with alpha not a digit, so we could get rid of .tar.gz etc

Parameters

- **name** (str | Path) –
- **return_ext** (bool) –

Return type
str | tuple[str, str]

`datalad.utils.find_files(regex, topdir='.', exclude=None, exclude_vcs=True, exclude_datalad=False, dirs=False)`

Generator to find files matching regex

Parameters

- **regex** (*string*) –
- **exclude** (*string, optional*) – Matches to exclude
- **exclude_vcs** (bool) – If True, excludes commonly known VCS subdirectories. If string, used as regex to exclude those files (regex: *'⋀(?:git|gitattributes|svn|bzr|hg)(?:/|$)'*)
- **exclude_datalad** (bool) – If True, excludes files known to be datalad meta-data files (e.g. under .datalad/ subdirectory) (regex: *'⋀(?:datalad)(?:/|$)'*)
- **topdir** (*string, optional*) – Directory where to search
- **dirs** (*bool, optional*) – Whether to match directories as well as files

Return type
Iterator[str]

`datalad.utils.generate_chunks(container, size)`

Given a container, generate chunks from it with size up to *size*

Parameters

- **container** (`list[TypeVar(T)]`) –
- **size** (`int`) –

Return type
`Iterator[list[TypeVar(T)]]`

`datalad.utils.generate_file_chunks(files, cmd=None)`

Given a list of files, generate chunks of them to avoid exceeding cmdline length

Parameters

- **files** (*list of str*) –
- **cmd** (*str or list of str, optional*) – Command to account for as well

Return type
Iterator[list[str]]

`datalad.utils.get_dataset_root(path)`

Return the root of an existent dataset containing a given path

The root path is returned in the same absolute or relative form as the input argument. If no associated dataset exists, or the input path doesn't exist, None is returned.

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

Parameters
path (*Path-like*) –

Return type
str or None

datalad.utils.**get_encoding_info**()

Return a dictionary with various encoding/locale information

Return type
dict[str, str]

datalad.utils.**get_envvars_info**()

Return type
dict[str, str]

datalad.utils.**get_home_envvars**(*new_home*)

Return dict with env variables to be adjusted for a new HOME

Only variables found in current os.environ are adjusted.

Parameters
new_home (*str or Path*) – New home path, in native to OS "schema"

Return type
dict[str, str]

datalad.utils.**get_ipython_shell**()

Detect if running within IPython and returns its *ip* (shell) object

Returns None if not under ipython (no *get_ipython* function)

Return type
Optional[Any]

datalad.utils.**get_linux_distribution**()

Compatibility wrapper for {platform,distro}.linux_distribution().

Return type
tuple[str, str, str]

datalad.utils.**get_logfilename**(*dspath*, *cmd='datalad'*)

Return a filename to use for logging under a dataset/repository

directory would be created if doesn't exist, but dspath must exist and be a directory

Parameters

- **dspath** (str | Path) –
- **cmd** (str) –

Return type
str

datalad.utils.**get_open_files**(*path*, *log_open=False*)

Get open files under a path

Note: This function is very slow on Windows.

Parameters

- **path** (*str*) – File or directory to check for open files under
- **log_open** (*bool or int*) – If set - logger level to use

Returns
path : pid

Return type
dict

datalad.utils.**get_path_prefix**(*path*, *pwd=None*)

Get path prefix (for current directory)

Returns relative path to the topdir, if we are under topdir, and if not absolute path to topdir. If *pwd* is not specified - current directory assumed

Parameters

- **path** (str | Path) –
- **pwd** (Optional[str]) –

Return type
str

datalad.utils.**get_sig_param_names**(*f*, *kinds*)

A helper to selectively return parameters from inspect.signature.

inspect.signature is the ultimate way for introspecting callables. But its interface is not so convenient for a quick selection of parameters (AKA arguments) of desired type or combinations of such. This helper should make it easier to retrieve desired collections of parameters.

Since often it is desired to get information about multiple specific types of parameters, *kinds* is a list, so in a single invocation of *signature* and looping through the results we can obtain all information.

Parameters

- **f** (*callable*) –
- **kinds** (*tuple with values from {'pos_any', 'pos_only', 'kw_any', 'kw_only', 'any'}*) – Is a list of what kinds of args to return in result (tuple). Each element should be one of: 'any_pos' - positional or keyword which could be used positionally. 'kw_only' - keyword only (cannot be used positionally) arguments, 'any_kw` - any keyword (could be a positional which could be used as a keyword), *any* – any type from the above.

Returns
Each element is a list of parameters (names only) of that "kind".

Return type
tuple

datalad.utils.**get_suggestions_msg**(*values*, *known*, *sep='\\n '*)

Return a formatted string with suggestions for values given the known ones

Parameters

- **values** (Optional[str | Iterable[str]]) –
- **known** (str) –
- **sep** (str) –

Return type
str

datalad.utils.**get_tempfile_kwargs**(*tkwargs=None*, *prefix=''*, *wrapped=None*)

Updates kwargs to be passed to tempfile. calls depending on env vars

Parameters

- **tkwargs** (Optional[dict[str, Any]]) –

- **prefix** (str) –
- **wrapped** (Optional[Callable]) –

Return type
dict[str, Any]

datalad.utils.**get_timestamp_suffix**(*time_=None*, *prefix='-'*)

Return a time stamp (full date and time up to second)

primarily to be used for generation of log files names

Parameters

- **time_** (int | time.struct_time | None) –
- **prefix** (str) –

Return type
str

datalad.utils.**get_trace**(*edges*, *start*, *end*, *trace=None*)

Return the trace/path to reach a node in a tree.

Parameters

- **edges** (*sequence(2-tuple)*) – The tree given by a sequence of edges (parent, child) tuples. The nodes can be identified by any value and data type that supports the '==' operation.
- **start** (TypeVar(T)) – Identifier of the start node. Must be present as a value in the parent location of an edge tuple in order to be found.
- **end** (TypeVar(T)) – Identifier of the target/end node. Must be present as a value in the child location of an edge tuple in order to be found.
- **trace** (*list*) – Mostly useful for recursive calls, and used internally.

Returns
Returns a list with the trace to the target (the starts and the target are not included in the trace, hence if start and end are directly connected an empty list is returned), or None when no trace to the target can be found, or start and end are identical.

Return type
None or list

datalad.utils.**get_wrapped_class**(*wrapped*)

Determine the command class a wrapped __call__ belongs to

Parameters
wrapped (Callable) –

Return type
type

datalad.utils.**getargspec**(*func*, *, *include_kwonlyargs=False*)

Compat shim for getargspec deprecated in python 3.

The main difference from inspect.getargspec (and inspect.getfullargspec for that matter) is that by using inspect.signature we are providing correct args/defaults for functools.wraps'ed functions.

include_kwonlyargs option was added to centralize getting all args, even the ones which are kwonly (follow the `*,`).

For internal use and not advised for use in 3rd party code. Please use inspect.signature directly.

Parameters

- **func** (Callable[..., Any]) –
- **include_kwonlyargs** (bool) –

Return type
ArgSpecFake

datalad.utils.**getpwd**()

Try to return a CWD without dereferencing possible symlinks

This function will try to use PWD environment variable to provide a current working directory, possibly with some directories along the path being symlinks to other directories. Unfortunately, PWD is used/set only by the shell and such functions as *os.chdir* and *os.getcwd* nohow use or modify it, thus *os.getcwd()* returns path with links dereferenced.

While returning current working directory based on PWD env variable we verify that the directory is the same as *os.getcwd()* after resolving all symlinks. If that verification fails, we fall back to always use *os.getcwd()*.

Initial decision to either use PWD env variable or os.getcwd() is done upon the first call of this function.

Return type
str

datalad.utils.**guard_for_format**(*arg*)

Replace { and } with {{ and }}

To be used in cases if arg is not expected to have provided by user .format() placeholders, but 'arg' might become a part of a composite passed to .format(), e.g. via 'Run'

Parameters
arg (str) –

Return type
str

datalad.utils.**import_module_from_file**(*modpath*, *pkg=None*, *log=<bound method Logger.debug of <Logger datalad.utils (INFO)>>*)

Import provided module given a path

TODO: - RF/make use of it in pipeline.py which has similar logic - join with import_modules above?

Parameters

- **pkg** (*module, optional*) – If provided, and modpath is under pkg.__path__, relative import will be used
- **modpath** (str) –
- **log** (Callable[[str], Any]) –

Return type
ModuleType

datalad.utils.**import_modules**(*modnames*, *pkg*, *msg='Failed to import {module}'*, *log=<bound method Logger.debug of <Logger datalad.utils (INFO)>>*)

Helper to import a list of modules without failing if N/A

Parameters

- **modnames** (*list of str*) – List of module names to import
- **pkg** (*str*) – Package under which to import

- **msg** (*str, optional*) – Message template for .format() to log at DEBUG level if import fails. Keys {module} and {package} will be provided and ': {exception}' appended
- **log** (*callable, optional*) – Logger call to use for logging messages

Return type
list[ModuleType]

datalad.utils.**is_explicit_path**(*path*)

Return whether a path explicitly points to a location

Any absolute path, or relative path starting with either '../' or './' is assumed to indicate a location on the filesystem. Any other path format is not considered explicit.

Parameters
path (str | Path) –

Return type
bool

datalad.utils.**is_interactive**()

Return True if all in/outs are open and tty.

Note that in a somewhat abnormal case where e.g. stdin is explicitly closed, and any operation on it would raise a *ValueError("I/O operation on closed file")* exception, this function would just return False, since the session cannot be used interactively.

Return type
bool

datalad.utils.**join_cmdline**(*args*)

Join command line args into a string using quote_cmdlinearg

Parameters
args (Iterable[str]) –

Return type
str

datalad.utils.**knows_annex**(*path*)

Returns whether at a given path there is information about an annex

It is just a thin wrapper around GitRepo.is_with_annex() classmethod which also checks for *path* to exist first.

This includes actually present annexes, but also uninitialized ones, or even the presence of a remote annex branch.

Parameters
path (str | Path) –

Return type
bool

datalad.utils.**line_profile**(*func*)

Q&D helper to line profile the function and spit out stats

Parameters
func (Callable[[ParamSpec(P)], TypeVar(T)]) –

Return type
Callable[[ParamSpec(P)], TypeVar(T)]

datalad.utils.**lmtime**(*filepath*, *mtime*)

Set mtime for files, while not de-referencing symlinks.

To overcome absence of os.lutime

Works only on linux and OSX ATM

Parameters

- **filepath** (str | Path) –
- **mtime** (int | float) –

Return type
None

datalad.utils.**lock_if_required**(*lock_required*, *lock*)

Acquired and released the provided lock if indicated by a flag

Parameters

- **lock_required** (bool) –
- **lock** (allocate_lock) –

Return type
Iterator[allocate_lock]

datalad.utils.**make_tempfile**(*content=None*, *wrapped=None*, ***tkwargs*)

Helper class to provide a temporary file name and remove it at the end (context manager)

Parameters

- **mkdir** (*bool, optional (default: False)*) – If True, temporary directory created using tempfile.mkdtemp()
- **content** (*str or bytes, optional*) – Content to be stored in the file created
- **wrapped** (*function, optional*) – If set, function name used to prefix temporary file name
- ****tkwargs** – All other arguments are passed into the call to tempfile.mk{,d}temp(), and resultant temporary filename is passed as the first argument into the function t. If no 'prefix' argument is provided, it will be constructed using module and function names ('.' replaced with '_').
- **set** (*To change the used directory without providing keyword argument 'dir'*) –
- **DATALAD_TESTS_TEMP_DIR.** –

Return type
Iterator[str]

Examples

```
>>> from os.path import exists
>>> from datalad.utils import make_tempfile
>>> with make_tempfile() as fname:
...    k = open(fname, 'w').write('silly test')
>>> assert not exists(fname)  # was removed
```

```
>>> with make_tempfile(content="blah") as fname:
...    assert open(fname).read() == "blah"
```

Parameters
tkwargs (Any) –

datalad.utils.**map_items**(*func*, *v*)

A helper to apply *func* to all elements (keys and values) within dict

No type checking of values passed to func is done, so *func* should be resilient to values which it should not handle

Initial usecase - apply_recursive(url_fragment, ensure_unicode)

datalad.utils.**md5sum**(*filename*)

Compute an MD5 sum for the given file

Parameters
filename (str | Path) –

Return type
str

datalad.utils.**never_fail**(*f*)

Assure that function never fails – all exceptions are caught

Returns *None* if function fails internally.

Parameters
f (Callable[[ParamSpec(P)], TypeVar(T)]) –

Return type
Callable[[ParamSpec(P)], Optional[TypeVar(T)]]

datalad.utils.**not_supported_on_windows**(*msg=None*)

A little helper to be invoked to consistently fail whenever functionality is not supported (yet) on Windows

Parameters
msg (Optional[str]) –

Return type
None

datalad.utils.**nothing_cm**()

Just a dummy cm to programmically switch context managers

Return type
Iterator[None]

datalad.utils.**obtain_write_permission**(*path*)

Obtains write permission for *path* and returns previous mode if a change was actually made.

Parameters
path (*Path*) – path to try to obtain write permission for

Returns
previous mode of *path* as return by stat().st_mode if a change in permission was actually necessary, *None* otherwise.

Return type
int or None

datalad.utils.**open_r_encdetect**(*fname*, *readahead=1000*)

Return a file object in read mode with auto-detected encoding

This is helpful when dealing with files of unknown encoding.

Parameters

- **readahead** (*int, optional*) – How many bytes to read for guessing the encoding type. If negative - full file will be read
- **fname** (str | Path) –

Return type
IO[str]

datalad.utils.**optional_args**(*decorator*)

allows a decorator to take optional positional and keyword arguments. Assumes that taking a single, callable, positional argument means that it is decorating a function, i.e. something like this:

```
@my_decorator
def function(): pass
```

Calls decorator with decorator(f, **args*, ***kwargs*)

datalad.utils.**partition**(*items*, *predicate=<class 'bool'>*)

Partition *items* by *predicate*.

Parameters

- **items** (*iterable*) –
- **predicate** (*callable*) – A function that will be mapped over each element in *items*. The elements will partitioned based on whether the return value is false or true.

Return type
tuple[Iterator[TypeVar(T)], Iterator[TypeVar(T)]]

Returns

- *A tuple with two generators, the first for 'false' items and the second for*
- *'true' ones.*

Notes

Taken from Peter Otten's snippet posted at https://nedbatchelder.com/blog/201306/filter_a_list_into_two_parts.html

`datalad.utils.path_is_subpath(path, prefix)`

Return True if path is a subpath of prefix

It will return False if path == prefix.

Parameters

- **path** (*str*) –
- **prefix** (*str*) –

Return type
bool

`datalad.utils.path_startswith(path, prefix)`

Return True if path starts with prefix path

Parameters

- **path** (*str*) –
- **prefix** (*str*) –

Return type
bool

`datalad.utils.posix_relpath(path, start=None)`

Behave like os.path.relpath, but always return POSIX paths…

on any platform.

Parameters

- **path** (str | Path) –
- **start** (Optional[str | Path]) –

Return type
str

`datalad.utils.quote_cmdlinearg(arg)`

Perform platform-appropriate argument quoting

Parameters
arg (str) –

Return type
str

`datalad.utils.read_csv_lines(fname, dialect=None, readahead=16384, **kwargs)`

A generator of dict records from a CSV/TSV

Automatically guesses the encoding for each record to convert to UTF-8

Parameters

- **fname** (*str*) – Filename
- **dialect** (*str, optional*) – Dialect to specify to csv.reader. If not specified – guessed from the file, if fails to guess, "excel-tab" is assumed

- **readahead** (*int, optional*) – How many bytes to read from the file to guess the type
- ****kwargs** (Any) – Passed to *csv.reader*

Return type
Iterator[dict[str, str]]

datalad.utils.**read_file**(*fname*, *decode=True*)

A helper to read file passing content via ensure_unicode

Parameters

- **decode** (*bool, optional*) – if False, no ensure_unicode and file content returned as bytes
- **fname** (str | Path) –

Return type
str | bytes

datalad.utils.**rmdir**(*path*, **args*, ***kwargs*)

os.rmdir with our optional checking for open files

Parameters

- **path** (str | Path) –
- **args** (Any) –
- **kwargs** (Any) –

Return type
None

datalad.utils.**rmtemp**(*f*, **args*, ***kwargs*)

Wrapper to centralize removing of temp files so we could keep them around

It will not remove the temporary file/directory if DATALAD_TESTS_TEMP_KEEP environment variable is defined

Parameters

- **f** (str | Path) –
- **args** (Any) –
- **kwargs** (Any) –

Return type
None

datalad.utils.**rmtree**(*path*, *chmod_files='auto'*, *children_only=False*, **args*, ***kwargs*)

To remove git-annex .git it is needed to make all files and directories writable again first

Parameters

- **path** (*Path or str*) – Path to remove
- **chmod_files** (*string or bool, optional*) – Whether to make files writable also before removal. Usually it is just a matter of directories to have write permissions. If 'auto' it would chmod files on windows by default
- **children_only** (*bool, optional*) – If set, all files and subdirectories would be removed while the path itself (must be a directory) would be preserved
- ***args** –

- ****kwargs** – Passed into shutil.rmtree call
- **args** (Any) –
- **kwargs** (Any) –

Return type
None

datalad.utils.**rotree**(*path*, *ro=True*, *chmod_files=True*)

To make tree read-only or writable

Parameters

- **path** (*string*) – Path to the tree/directory to chmod
- **ro** (*bool, optional*) – Whether to make it R/O (default) or RW
- **chmod_files** (*bool, optional*) – Whether to operate also on files (not just directories)

Return type
None

datalad.utils.**saved_generator**(*gen*)

Given a generator returns two generators, where 2nd one just replays

So the first one would be going through the generated items and 2nd one would be yielding saved items

Parameters
gen (Iterable[TypeVar(T)]) –

Return type
tuple[Iterator[TypeVar(T)], Iterator[TypeVar(T)]]

datalad.utils.**shortened_repr**(*value*, *l=30*)

Parameters

- **value** (Any) –
- **l** (int) –

Return type
str

datalad.utils.**slash_join**(*base*, *extension*)

Join two strings with a '/', avoiding duplicate slashes

If any of the strings is None the other is returned as is.

Parameters

- **base** (Optional[str]) –
- **extension** (Optional[str]) –

Return type
Optional[str]

datalad.utils.**split_cmdline**(*s*)

Perform platform-appropriate command line splitting.

Identical to *shlex.split()* on non-windows platforms.

Modified from https://stackoverflow.com/a/35900070

Parameters
s (str) –

Return type
list[str]

datalad.utils.**swallow_logs**(*new_level=None*, *file_=None*, *name='datalad'*)

Context manager to consume all logs.

Parameters

- **new_level** (str | int | None) –
- **file_** (str | Path | None) –
- **name** (str) –

Return type
Iterator[SwallowLogsAdapter]

datalad.utils.**swallow_outputs**()

Context manager to help consuming both stdout and stderr, and print()

stdout is available as cm.out and stderr as cm.err whenever cm is the yielded context manager. Internally uses temporary files to guarantee absent side-effects of swallowing into StringIO which lacks .fileno.

print mocking is necessary for some uses where sys.stdout was already bound to original sys.stdout, thus mocking it later had no effect. Overriding print function had desired effect

Return type
Iterator[*SwallowOutputsAdapter*]

datalad.utils.**todo_interface_for_extensions**(*f*)

Parameters
f (TypeVar(T)) –

Return type
TypeVar(T)

datalad.utils.**try_multiple**(*ntrials*, *exception*, *base*, *f*, **args*, ***kwargs*)

Call f multiple times making exponentially growing delay between the calls

datalad.utils.**try_multiple_dec**(*f*, *ntrials=None*, *duration=0.1*, *exceptions=None*, *increment_type=None*, *exceptions_filter=None*, *logger=None*)

Decorator to try function multiple times.

Main purpose is to decorate functions dealing with removal of files/directories and which might need a few seconds to work correctly on Windows which takes its time to release files/directories.

Parameters

- **ntrials** (*int, optional*) –
- **duration** (*float, optional*) – Seconds to sleep before retrying.
- **increment_type** (*{None, 'exponential'}*) – Note that if it is exponential, duration should typically be > 1.0 so it grows with higher power
- **exceptions** (*Exception or tuple of Exceptions, optional*) – Exception or a tuple of multiple exceptions, on which to retry

- **exceptions_filter** (*callable, optional*) – If provided, this function will be called with a caught exception instance. If function returns True - we will re-try, if False - exception will be re-raised without retrying.
- **logger** (*callable, optional*) – Logger to log upon failure. If not provided, will use stock logger at the level of 5 (heavy debug).
- **f** (Callable[P, T]) –

Return type
Callable[P, T]

datalad.utils.**unique**(*seq*, *key=None*, *reverse=False*)

Given a sequence return a list only with unique elements while maintaining order

This is the fastest solution. See https://www.peterbe.com/plog/uniqifiers-benchmark and http://stackoverflow.com/a/480227/1265472 for more information. Enhancement – added ability to compare for uniqueness using a key function

Parameters

- **seq** (Sequence[TypeVar(T)]) – Sequence to analyze
- **key** (*callable, optional*) – Function to call on each element so we could decide not on a full element, but on its member etc
- **reverse** (*bool, optional*) – If True, uniqueness checked in the reverse order, so that the later ones will take the order

Return type
list[TypeVar(T)]

datalad.utils.**unlink**(*f*)

'Robust' unlink. Would try multiple times

On windows boxes there is evidence for a latency of more than a second until a file is considered no longer "in-use". WindowsError is not known on Linux, and if IOError or any other exception is thrown then if except statement has WindowsError in it – NameError also see gh-2533

Parameters
f (str | Path) –

Return type
None

datalad.utils.**updated**(*d*, *update*)

Return a copy of the input with the 'update'

Primarily for updating dictionaries

Parameters

- **d** (dict[TypeVar(K), TypeVar(V)]) –
- **update** (dict[TypeVar(K), TypeVar(V)]) –

Return type
dict[TypeVar(K), TypeVar(V)]

datalad.utils.**with_pathsep**(*path*)

Little helper to guarantee that path ends with /

Parameters
path (str) –

Return type
`str`

datalad.version

datalad.support.gitrepo

Internal low-level interface to Git repositories

class datalad.support.gitrepo.**FetchInfo**

Bases: `dict`

dict that carries results of a fetch operation of a single head

Reduced variant of GitPython's RemoteProgress class

Original copyright:
Copyright (C) 2008, 2009 Michael Trier and contributors

Original license:
BSD 3-Clause "New" or "Revised" License

ERROR = 128

FAST_FORWARD = 64

FORCED_UPDATE = 32

HEAD_UPTODATE = 4

NEW_HEAD = 2

NEW_TAG = 1

REJECTED = 16

TAG_UPDATE = 8

class datalad.support.gitrepo.**GitAddOutput**

Bases: `TypedDict`

file: str

success: bool

class datalad.support.gitrepo.**GitProgress**(*done_future=None*, *encoding=None*)

Bases: *`WitlessProtocol`*

Reduced variant of GitPython's RemoteProgress class

Original copyright:
Copyright (C) 2008, 2009 Michael Trier and contributors

Original license:
BSD 3-Clause "New" or "Revised" License

Parameters

- **done_future** (`Optional[Any]`) –
- **encoding** (`Optional[str]`) –

BEGIN = 1

CHECKING_OUT = 256

COMPRESSING = 8

COUNTING = 4

DONE_TOKEN = 'done.'

END = 2

ENUMERATING = 512

FINDING_SOURCES = 128

OP_MASK = -4

RECEIVING = 32

RESOLVING = 64

STAGE_MASK = 3

TOKEN_SEPARATOR = ', '

WRITING = 16

connection_made(*transport*)

Parameters
transport (Popen) –

Return type
None

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

pipe_data_received(*fd*, *byts*)

Parameters

- fd (int) –
- byts (bytes) –

Return type
None

proc_err = True

process: Optional[Popen]

process_exited()

Return type
None

re_op_absolute = re.compile('(remote:)?([\\w\\s]+):\\s+()(\\d+)()(.*)')

re_op_relative = re.compile('(remote:)?([\\w\\s]+):\\s+(\\d+)%
\\((\\d+)/(\\d+)\\)(.*)')

class datalad.support.gitrepo.**GitRepo**(*path*, *runner=None*, *create=True*, *git_opts=None*, *repo=None*, *fake_dates=False*, *create_sanity_checks=True*, ***kwargs*)

Bases: GitRepo

Representation of a git repository

Parameters

- **path** (str) –
- **runner** (Optional[Any]) –
- **create** (bool) –
- **git_opts** (Optional[dict[str, Any]]) –
- **repo** (Optional[Any]) –
- **fake_dates** (bool) –
- **create_sanity_checks** (bool) –
- **kwargs** (Any) –

GIT_MIN_VERSION = '2.25'

add(*files*, *git=True*, *git_options=None*, *update=False*)

Adds file(s) to the repository.

Parameters

- **files** (*list*) – list of paths to add
- **git** (*bool*) – somewhat ugly construction to be compatible with AnnexRepo.add(); has to be always true.
- **update** (*bool*) –

  –update option for git-add. From git's manpage:
  Update the index just where it already has an entry matching <pathspec>. This removes as well as modifies index entries to match the working tree, but adds no new files.

  If no <pathspec> is given when –update option is used, all tracked files in the entire working tree are updated (old versions of Git used to limit the update to the current directory and its subdirectories).
- **git_options** (Optional[list[str]]) –

Returns
Of status dicts.

Return type
list

add_(*files*, *git=True*, *git_options=None*, *update=False*)

Like *add*, but returns a generator

Parameters

- **files** (list[str]) –
- **git** (bool) –
- **git_options** (Optional[list[str]]) –
- **update** (bool) –

Return type
`Iterator[GitAddOutput]`

`add_fake_dates(env)`

`add_remote(name, url, options=None)`

Register remote pointing to a url

Parameters

- **name** (`str`) –
- **url** (`str`) –
- **options** (`Optional[list[str]]`) –

Return type
`tuple[str, str]`

`property bare: bool`

Returns a bool indicating whether the repository is bare

Importantly, this is not reporting the configuration value of 'core.bare', in order to be usable at a stage where a Repo instance is not yet equipped with a ConfigManager. Instead, it is testing whether the repository path and its "dot_git" are identical. The value of 'core.bare' can be query from the ConfigManager in a fully initialized instance.

`checkout(name, options=None)`

Parameters

- **name** (`str`) –
- **options** (`Optional[list[str]]`) –

Return type
`None`

`cherry_pick(commit)`

Cherry pick *commit* to the current branch.

Parameters
commit (*str*) – A single commit.

Return type
`None`

`classmethod clone(url, path, *args, clone_options=None, **kwargs)`

Clone url into path

Provides workarounds for known issues (e.g. https://github.com/datalad/datalad/issues/785)

Parameters

- **url** (*str*) –
- **path** (*str*) –
- **clone_options** (*dict or list*) – Arbitrary options that will be passed on to the underlying call to *git-clone*. This may be a list of plain options or key-value pairs that will be converted to a list of plain options with *to_options*.
- **expect_fail** (*bool*) – Whether expect that command might fail, so error should be logged then at DEBUG level instead of ERROR

- **kwargs** (Any) – Passed to the Repo class constructor.
- **args** (Any) –

Return type
Self

commit(*msg=None*, *options=None*, *_datalad_msg=False*, *careless=True*, *files=None*, *date=None*, *index_file=None*)

Commit changes to git.

Parameters

- **msg** (*str, optional*) – commit-message
- **options** (*list of str, optional*) – cmdline options for git-commit
- **_datalad_msg** (*bool, optional*) – To signal that commit is automated commit by datalad, so it would carry the [DATALAD] prefix
- **careless** (*bool, optional*) – if False, raise when there's nothing actually committed; if True, don't care
- **files** (*list of str, optional*) – path(s) to commit
- **date** (*str, optional*) – Date in one of the formats git understands
- **index_file** (*str, optional*) – An alternative index to use

Return type
None

commit_exists(*commitish*)

Does *commitish* exist in the repo?

Parameters
commitish (*str*) – A commit or an object that can be dereferenced to one.

Return type
bool

property config

configure_fake_dates()

Configure repository to use fake dates.

Return type
None

property count_objects: dict[str, int]

return dictionary with count, size(in KiB) information of git objects

describe(*commitish=None*, ***kwargs*)

Quick and dirty implementation to call git-describe

Parameters

- **kwargs** (Union[str, bool, None, List[Union[str, bool, None]], Tuple[Union[str, bool, None], ...]]) – transformed to cmdline options for git-describe; see __init__ for description of the transformation
- **commitish** (Optional[str]) –

Return type
Optional[str]

diff(*fr*, *to*, *paths=None*, *untracked='all'*, *eval_submodule_state='full'*)

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

Parameters

- **fr** (*str or None*) – Revision specification (anything that Git understands). Passing *None* considers anything in the target state as new.
- **to** (*str or None*) – Revision specification (anything that Git understands), or None to compare to the state of the work tree.
- **paths** (*list or None*) – If given, limits the query to the specified paths. To query all paths specify *None*, not an empty list.
- **untracked** (*{'no', 'normal', 'all'}*) – If and how untracked content is reported when *to* is None: 'no': no untracked files are reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories.
- **eval_submodule_state** (*{'full', 'commit', 'no'}*) – If 'full' (the default), the state of a submodule is evaluated by considering all modifications, with the treatment of untracked files determined by *untracked*. If 'commit', the modification check is restricted to comparing the submodule's HEAD commit to the one recorded in the superdataset. If 'no', the state of the subdataset is not evaluated.

Returns

Each content item has an entry under a pathlib *Path* object instance pointing to its absolute path inside the repository (this path is guaranteed to be underneath *Repo.path*). Each value is a dictionary with properties:

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

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

Return type
dict

diffstatus(*fr*, *to*, *paths=None*, *untracked='all'*, *eval_submodule_state='full'*, *_cache=None*)

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

It supports an additional submodule evaluation state 'global'. If given, it will return a single 'modified' (vs. 'clean') state label for the entire repository, as soon as it can.

Parameters

- **fr** (Optional[str]) –
- **to** (Optional[str]) –
- **paths** (Optional[Sequence[str | PathLike[str]]]) –
- **untracked** (str) –
- **eval_submodule_state** (str) –
- **_cache** (Optional[dict]) –

Return type
dict[Path, dict[str, str]] | str

property dirty: bool

Is the repository dirty?

Note: This provides a quick answer when you simply want to know if there are any untracked changes or modifications in this repository or its submodules. For finer-grained control and more detailed reporting, use status() instead.

property fake_dates_enabled: bool

Is the repository configured to use fake dates?

fetch(*remote=None*, *refspec=None*, *all_=False*, *git_options=None*, ***kwargs*)

Fetches changes from a remote (or all remotes).

Parameters

- **remote** (*str, optional*) – name of the remote to fetch from. If no remote is given and *all_* is not set, the tracking branch is fetched.
- **refspec** (*str or list, optional*) – refspec(s) to fetch.
- **all** (*bool, optional*) – fetch all remotes (and all of their branches). Fails if *remote* was given.
- **git_options** (*list, optional*) – Additional command line options for git-fetch.
- **kwargs** (Option) – Deprecated. GitPython-style keyword argument for git-fetch. Will be appended to any git_options.
- **all_** (bool) –

Return type
list[FetchInfo]

fetch_(*remote=None*, *refspec=None*, *all_=False*, *git_options=None*)

Like *fetch*, but returns a generator

Parameters

- **remote** (Optional[str]) –
- **refspec** (str | list[str] | None) –
- **all_** (bool) –
- **git_options** (Optional[list[str]]) –

Return type
Iterator[FetchInfo]

format_commit(*fmt*, *commitish=None*)

Return *git show* output for *commitish*.

Parameters

- **fmt** (*str*) – A format string accepted by *git show*.
- **commitish** (*str, optional*) – Any commit identifier (defaults to "HEAD").

Return type
str or, if there are not commits yet, None.

gc(*allow_background=False*, *auto=False*)

Perform house keeping (garbage collection, repacking)

Parameters

- **allow_background** (bool) –
- **auto** (bool) –

Return type
None

get_active_branch()

Get the name of the active branch

Returns
Returns None if there is no active branch, i.e. detached HEAD, and the branch name otherwise.

Return type
str or None

get_branch_commits_(*branch=None*, *limit=None*, *stop=None*)

Return commit hexshas for a branch

Parameters

- **branch** (*str, optional*) – If not provided, assumes current branch
- **limit** (*None | 'left-only', optional*) – Limit which commits to report. If None – all commits (merged or not), if 'left-only' – only the commits from the left side of the tree upon merges
- **stop** (*str, optional*) – hexsha of the commit at which stop reporting (matched one is not reported either)

Yields
str

Return type
Iterator[str]

get_branches()

Get all branches of the repo.

Returns
Names of all branches of this repository.

Return type
[str]

get_commit_date(*branch=None*, *date='authored'*)

Get the date stamp of the last commit (in a branch or head otherwise)

Parameters

- **date** (*{'authored', 'committed'}*) – Which date to return. "authored" will be the date shown by "git show" and the one possibly specified via –date to *git commit*
- **branch** (Optional[str]) –

Returns
None if no commit

Return type
int or None

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

Get identifier and type information from repository content.

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

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

Parameters

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

Returns

Each content item has an entry under a pathlib *Path* object instance pointing to its absolute path inside the repository (this path is guaranteed to be underneath *Repo.path*). Each value is a dictionary with properties:

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

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

Return type
dict

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

get_corresponding_branch(*branch=None*)

Always returns None, a plain GitRepo has no managed branches

Parameters
branch (`Optional[Any]`) –

Return type
`Optional[str]`

get_files(*branch=None*)

Get a list of files in git.

Lists the files in the (remote) branch.

Parameters
branch (*str*) – Name of the branch to query. Default: active branch.

Returns
list of files.

Return type
[str]

`get_git_attributes()`

Query gitattributes which apply to top level directory

It is a thin compatibility/shortcut wrapper around more versatile get_gitattributes which operates on a list of paths and returns a dictionary per each path

Returns
a dictionary with attribute name and value items relevant for the top ('.') directory of the repository, and thus most likely the default ones (if not overwritten with more rules) for all files within repo.

Return type
dict

`static get_git_dir(repo)`

figure out a repo's gitdir

'.git' might be a directory, a symlink or a file

Note: This method is likely to get deprecated, please use GitRepo.dot_git instead! That one's not static, but it's cheaper and you should avoid not having an instance of a repo you're working on anyway. Note, that the property in opposition to this method returns an absolute path.

Parameters
repo (*path or Repo instance*) – currently expected to be the repos base dir

Returns
relative path to the repo's git dir; So, default would be ".git"

Return type
str

`get_gitattributes(path, index_only=False)`

Query gitattributes for one or more paths

Parameters

- **path** (*path or list*) – Path(s) to query. Paths may be relative or absolute.
- **index_only** (*bool*) – Flag whether to consider only gitattribute setting that are reflected in the repository index, not just in the work tree content.

Returns
Each key is a queried path (always relative to the repository root), each value is a dictionary with attribute name and value items. Attribute values are either True or False, for set and unset attributes, or are the literal attribute value.

Return type
dict

`get_hexsha(commitish=None, short=False)`

Return a hexsha for a given commitish.

Parameters

- **commitish** (*str, optional*) – Any identifier that refers to a commit (defaults to "HEAD").

- **short** (*bool, optional*) – Return the abbreviated form of the hexsha.

Return type
str or, if no commitish was given and there are no commits yet, None.

Raises
ValueError – If a commitish was given, but no corresponding commit could be determined.

get_indexed_files()

Get a list of files in git's index

Returns
list of paths rooting in git's base dir

Return type
list

get_last_commit_hexsha(*files*)

Return the hash of the last commit the modified any of the given paths

Parameters
files (list[str]) –

Return type
Optional[str]

get_merge_base(*commitishes*)

Get a merge base hexsha

Parameters
commitishes (*str or list of str*) – List of commitishes (branches, hexshas, etc) to determine the merge base of. If a single value provided, returns merge_base with the current branch.

Returns
If no merge-base for given commits, or specified treeish doesn't exist, None returned

Return type
str or None

get_remote_branches()

Get all branches of all remotes of the repo.

Returns
Names of all remote branches.

Return type
[str]

get_remote_url(*name*, *push=False*)

Get the url of a remote.

Reads the configuration of remote *name* and returns its url or None, if there is no url configured.

Parameters

- **name** (*str*) – name of the remote
- **push** (*bool*) – if True, get the pushurl instead of the fetch url.

Return type
Optional[str]

get_remotes(*with_urls_only=False*)

Get known remotes of the repository

Parameters
with_urls_only (*bool, optional*) – return only remotes which have urls

Returns
remotes – List of names of the remotes

Return type
list of str

get_revisions(*revrange=None*, *fmt='%H'*, *options=None*)

Return list of revisions in *revrange*.

Parameters

- **revrange** (*str or list of str or None, optional*) – Revisions or revision ranges to walk. If None, revision defaults to HEAD unless a revision-modifying option like *–all* or *–branches* is included in *options*.
- **fmt** (*string, optional*) – Format accepted by *–format* option of *git log*. This should not contain new lines because the output is split on new lines.
- **options** (*list of str, optional*) – Options to pass to *git log*. This should not include *–format*.

Return type
List of revisions (str), formatted according to *fmt*.

get_staged_paths()

Returns a list of any stage repository path(s)

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

Return type
`list[str]`

get_submodules(*sorted_=True*, *paths=None*)

Return list of submodules.

Parameters

- **sorted** (*bool, optional*) – Sort submodules by path name.
- **paths** (*list(pathlib.PurePath), optional*) – Restrict submodules to those under *paths*.
- **sorted_** (bool) –

Return type
list[dict]

Returns

- List of submodule namedtuples if *compat* is true or otherwise a list
- of dictionaries as returned by *get_submodules_*.

get_submodules_(*paths=None*)

Yield submodules in this repository.

Parameters

paths (*list(pathlib.PurePath), optional*) – Restrict submodules to those under *paths*. Paths must be relative to the resolved repository root, and must be normed to match the reporting done by Git, i.e. no parent dir components (ala "some/../this").

Return type

Iterator[dict]

Returns

- *A generator that yields a dictionary with information for each*
- *submodule.*

get_tags(*output=None*)

Get list of tags

Parameters

output (*str, optional*) – If given, limit the return value to a list of values matching that particular key of the tag properties.

Returns

Each item is a dictionary with information on a tag. At present this includes 'hexsha', and 'name', where the latter is the string label of the tag, and the former the hexsha of the object the tag is attached to. The list is sorted by the creator date (committer date for lightweight tags and tagger date for annotated tags), with the most recent commit being the last element.

Return type

list

classmethod get_toppath(*path, follow_up=True, git_options=None*)

Return top-level of a repository given the path.

Parameters

- **follow_up** (*bool*) – If path has symlinks – they get resolved by git. If follow_up is True, we will follow original path up until we hit the same resolved path. If no such path found, resolved one would be returned.
- **git_options** (*list of str*) – options to be passed to the git rev-parse call
- **repository.** (*Return None if no parent directory contains a git*) –
- **path** (str) –

Return type

Optional[str]

get_tracking_branch(*branch=None, remote_only=False*)

Get the tracking branch for *branch* if there is any.

Parameters

- **branch** (*str*) – local branch to look up. If none is given, active branch is used.
- **remote_only** (*bool*) – Don't return a value if the upstream remote is set to "." (meaning this repository).

Returns

(remote or None, refspec or None) of the tracking branch

Return type

tuple

git_version = None

is_ancestor(*reva*, *revb*)

Is *reva* an ancestor of *revb*?

Parameters

- **reva** (*str*) – Revisions.
- **revb** (*str*) – Revisions.

Return type
bool

is_valid_git()

Returns whether the underlying repository appears to be still valid

Note, that this almost identical to the classmethod is_valid_repo(). However, if we are testing an existing instance, we can save Path object creations. Since this testing is done a lot, this is relevant. Creation of the Path objects in is_valid_repo() takes nearly half the time of the entire function.

Also note, that this method is bound to an instance but still class-dependent, meaning that a subclass cannot simply overwrite it. This is particularly important for the call from within __init__(), which in turn is called by the subclasses' __init__. Using an overwrite would lead to the wrong thing being called.

Return type
bool

classmethod is_valid_repo(*path*)

Returns if a given path points to a git repository

Parameters
path (str) –

Return type
bool

is_with_annex()

Report if GitRepo (assumed) has (remotes with) a git-annex branch

Return type
bool

merge(*name*, *options=None*, *msg=None*, *allow_unrelated=False*, ***kwargs*)

Parameters

- **name** (str) –
- **options** (Optional[list[str]]) –
- **msg** (Optional[str]) –
- **allow_unrelated** (bool) –
- **kwargs** (Any) –

Return type
None

precommit()

Perform pre-commit maintenance tasks

Return type
None

push(*remote=None*, *refspec=None*, *all_remotes=False*, *all_=False*, *git_options=None*, ***kwargs*)

Push changes to a remote (or all remotes).

If remote and refspec are specified, and remote has *remote.{remote}.datalad-push-default-first* configuration variable set (e.g. by *create-sibling-github*), we will first push the first refspec separately to possibly ensure that the first refspec is chosen by remote as the "default branch". See https://github.com/datalad/datalad/issues/4997 Upon successful push if this variable was set in the local git config, we unset it, so subsequent pushes would proceed normally.

Parameters

- **remote** (*str, optional*) – name of the remote to push to. If no remote is given and *all_* is not set, the tracking branch is pushed.
- **refspec** (*str or list, optional*) – refspec(s) to push.
- **all** (*bool, optional*) – push to all remotes. Fails if *remote* was given.
- **git_options** (*list, optional*) – Additional command line options for git-push.
- **kwargs** (Option) – Deprecated. GitPython-style keyword argument for git-push. Will be appended to any git_options.
- **all_remotes** (bool) –
- **all_** (bool) –

Return type
list[PushInfo]

push_(*remote=None*, *refspec=None*, *all_=False*, *git_options=None*)

Like *push*, but returns a generator

Parameters

- **remote** (Optional[str]) –
- **refspec** (str | list[str] | None) –
- **all_** (bool) –
- **git_options** (Optional[list[str]]) –

Return type
Iterator[PushInfo]

remove(*files*, *recursive=False*, ***kwargs*)

Remove files.

Calls git-rm.

Parameters

- **files** (*list of str*) – list of paths to remove
- **recursive** (*False*) – whether to allow recursive removal from subdirectories
- **kwargs** (Union[str, bool, None, List[Union[str, bool, None]], Tuple[Union[str, bool, None], ...]]) – see *__init__*

Returns
list of successfully removed files.

Return type
[str]

remove_branch(*branch*)

Parameters
branch (str) –

Return type
None

remove_remote(*name*)

Remove existing remote

Parameters
name (str) –

Return type
None

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

Save dataset content.

Parameters

- **message** (*str or None*) – A message to accompany the changeset in the log. If None, a default message is used.
- **paths** (*list or None*) – Any content with path matching any of the paths given in this list will be saved. Matching will be performed against the dataset status (GitRepo.status()), or a custom status provided via *_status*. If no paths are provided, ALL non-clean paths present in the repo status or *_status* will be saved.
- **_status** (*dict or None*) – If None, Repo.status() will be queried for the given *ds*. If a dict is given, its content will be used as a constraint. For example, to save only modified content, but no untracked content, set *paths* to None and provide a *_status* that has no entries for untracked content.
- ****kwargs** (Any) – Additional arguments that are passed to underlying Repo methods. Supported:
  - git : bool (passed to Repo.add()
  - eval_submodule_state : {'full', 'commit', 'no'} passed to Repo.status()
  - untracked : {'no', 'normal', 'all'} - passed to Repo.status()
  - amend : bool (passed to GitRepo.commit)

Return type
list[dict]

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

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

Parameters

- **message** (Optional[str]) –
- **paths** (Optional[list[Path]]) –
- **_status** (Optional[dict[Path, dict[str, str]]]) –
- **kwargs** (Any) –

Return type
`Iterator[dict]`

`set_gitattributes`(*attrs*, *attrfile='.gitattributes'*, *mode='a'*)

Set gitattributes

By default appends additional lines to *attrfile*. Note, that later lines in *attrfile* overrule earlier ones, which may or may not be what you want. Set *mode* to 'w' to replace the entire file by what you provided in *attrs*.

Parameters

- **attrs** (*list*) – Each item is a 2-tuple, where the first element is a path pattern, and the second element is a dictionary with attribute key/value pairs. The attribute dictionary must use the same semantics as those returned by *get_gitattributes()*. Path patterns can use absolute paths, in which case they will be normalized relative to the directory that contains the target .gitattributes file (see *attrfile*).
- **attrfile** (*path*) – Path relative to the repository root of the .gitattributes file the attributes shall be set in.
- **mode** (*str*) – 'a' to append .gitattributes, 'w' to replace it

Return type
None

`set_remote_url`(*name*, *url*, *push=False*)

Set the URL a remote is pointing to

Sets the URL of the remote *name*. Requires the remote to already exist.

Parameters

- **name** (*str*) – name of the remote
- **url** (*str*) –
- **push** (*bool*) – if True, set the push URL, otherwise the fetch URL

Return type
`None`

`status`(*paths=None*, *untracked='all'*, *eval_submodule_state='full'*)

Simplified *git status* equivalent.

Parameters

- **paths** (*list or None*) – If given, limits the query to the specified paths. To query all paths specify *None*, not an empty list. If a query path points into a subdataset, a report is made on the subdataset record within the queried dataset only (no recursion).
- **untracked** (*{'no', 'normal', 'all'}*) – If and how untracked content is reported: 'no': no untracked files are reported; 'normal': untracked files and entire untracked directories are reported as such; 'all': report individual files even in fully untracked directories.
- **eval_submodule_state** (*{'full', 'commit', 'no'}*) – If 'full' (the default), the state of a submodule is evaluated by considering all modifications, with the treatment of untracked files determined by *untracked*. If 'commit', the modification check is restricted to comparing the submodule's HEAD commit to the one recorded in the superdataset. If 'no', the state of the subdataset is not evaluated.

Returns

Each content item has an entry under a pathlib *Path* object instance pointing to its absolute path inside the repository (this path is guaranteed to be underneath *Repo.path*). Each value is a dictionary with properties:

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

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

Return type
dict

tag(*tag*, *message=None*, *commit=None*, *options=None*)

Tag a commit

Parameters

- **tag** (*str*) – Custom tag label. Must be a valid tag name.
- **message** (*str, optional*) – If provided, adds ['-m', <message>] to the list of *git tag* arguments.
- **commit** (*str, optional*) – If provided, will be appended as last argument to the *git tag* call, and can be used to identify the commit that shall be tagged, if not HEAD.
- **options** (*list, optional*) – Additional command options, inserted prior a potential *commit* argument.

Return type
None

property untracked_files: list[str]

Legacy interface, do not use! Use the status() method instead.

Despite its name, it also reports on untracked datasets, and yields their names with trailing path separators.

update_ref(*ref*, *value*, *oldvalue=None*, *symbolic=False*)

Update the object name stored in a ref "safely".

Just a shim for *git update-ref* call if not symbolic, and *git symbolic-ref* if symbolic

Parameters

- **ref** (*str*) – Reference, such as *ref/heads/BRANCHNAME* or HEAD.
- **value** (*str*) – Value to update to, e.g. hexsha of a commit when updating for a branch ref, or branch ref if updating HEAD
- **oldvalue** (*str*) – Value to update from. Safeguard to be verified by git. This is only valid if *symbolic* is not True.
- **symbolic** (*None*) – To instruct if ref is symbolic, e.g. should be used in case of ref=HEAD

Return type
None

update_remote(*name=None*, *verbose=False*)

Parameters

- **name** (Optional[str]) –
- **verbose** (bool) –

Return type
None

class datalad.support.gitrepo.**PushInfo**

Bases: dict

dict that carries results of a push operation of a single head

Reduced variant of GitPython's RemoteProgress class

Original copyright:
Copyright (C) 2008, 2009 Michael Trier and contributors

Original license:
BSD 3-Clause "New" or "Revised" License

DELETED = 64

ERROR = 1024

FAST_FORWARD = 256

FORCED_UPDATE = 128

NEW_HEAD = 2

NEW_TAG = 1

NO_MATCH = 4

REJECTED = 8

REMOTE_FAILURE = 32

REMOTE_REJECTED = 16

UP_TO_DATE = 512

class datalad.support.gitrepo.**StdOutCaptureWithGitProgress**(*done_future=None*, *encoding=None*)

Bases: *GitProgress*

Parameters

- **done_future** (Optional[Any]) –
- **encoding** (Optional[str]) –

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

proc_out = True

process: Optional[Popen]

datalad.support.gitrepo.**normalize_path**(*func*)

Decorator to provide unified path conversion for a single file

Unlike normalize_paths, intended to be used for functions dealing with a single filename at a time

Note: This is intended to be used within the repository classes and therefore returns a class method!

The decorated function is expected to take a path at first positional argument (after 'self'). Additionally the class *func* is a member of, is expected to have an attribute 'path'.

Parameters
func (Callable[[_WithPath, str, ParamSpec(P)], TypeVar(T)]) –

Return type
Callable[[_WithPath, str, ParamSpec(P)], TypeVar(T)]

datalad.support.gitrepo.**normalize_paths**(*func*, *match_return_type=True*, *map_filenames_back=False*, *serialize=False*)

Decorator to provide unified path conversions.

Note: This is intended to be used within the repository classes and therefore returns a class method!

The decorated function is expected to take a path or a list of paths at first positional argument (after 'self'). Additionally the class *func* is a member of, is expected to have an attribute 'path'.

Accepts either a list of paths or a single path in a str. Passes a list to decorated function either way, but would return based on the value of match_return_type and possibly input argument.

If a call to the wrapped function includes normalize_path and it is False no normalization happens for that function call (used for calls to wrapped functions within wrapped functions, while possible CWD is within a repository)

Parameters

- **match_return_type** (*bool, optional*) – If True, and a single string was passed in, it would return the first element of the output (after verifying that it is a list of length 1). It makes easier to work with single files input.
- **map_filenames_back** (*bool, optional*) – If True and returned value is a dictionary, it assumes to carry entries one per file, and then filenames are mapped back to as provided from the normalized (from the root of the repo) paths
- **serialize** (*bool, optional*) – Loop through files giving only a single one to the function one at a time. This allows to simplify implementation and interface to annex commands which do not take multiple args in the same call (e.g. checkpresentkey)

datalad.support.gitrepo.**to_options**(*split_single_char_options=True*, ***kwargs*)

Transform keyword arguments into a list of cmdline options

Imported from GitPython.

Original copyright:
Copyright (C) 2008, 2009 Michael Trier and contributors

Original license:
BSD 3-Clause "New" or "Revised" License

Parameters

- **split_single_char_options** (*bool*) –
- **kwargs** (Union[str, bool, None, List[Union[str, bool, None]], Tuple[Union[str, bool, None], ...]]) –

Return type
list

datalad.support.annexrepo

Interface to git-annex by Joey Hess.

For further information on git-annex see https://git-annex.branchable.com/.

class datalad.support.annexrepo.**AnnexInitOutput**(*done_future=None*, *encoding=None*)

Bases: *WitlessProtocol*, AssemblingDecoderMixIn

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

pipe_data_received(*fd*, *byts*)

proc_err = True

proc_out = True

process: Optional[subprocess.Popen]

class datalad.support.annexrepo.**AnnexJsonProtocol**(*done_future=None*, *total_nbytes=None*)

Bases: *WitlessProtocol*

Subprocess communication protocol for *annex ... –json* commands

Importantly, parsed JSON content is returned as a result, not string output.

This protocol also handles git-annex's JSON-style progress reporting.

add_to_output(*json_object*)

connection_made(*transport*)

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

pipe_data_received(*fd*, *data*)

proc_err = True

proc_out = True

process: Optional[subprocess.Popen]

process_exited()

class datalad.support.annexrepo.**AnnexRepo**(**args*, ***kwargs*)

Bases: *GitRepo*, RepoInterface

Representation of an git-annex repository.

Paths given to any of the class methods will be interpreted as relative to PWD, in case this is currently beneath AnnexRepo's base dir (*self.path*). If PWD is outside of the repository, relative paths will be interpreted as relative to *self.path*. Absolute paths will be accepted either way.

GIT_ANNEX_MIN_VERSION = '8.20200309'

WEB_UUID = '00000000-0000-0000-0000-000000000001'

add(*files*, *git=None*, *backend=None*, *options=None*, *jobs=None*, *git_options=None*, *annex_options=None*, *update=False*)

Add file(s) to the repository.

Parameters

- **files** (*list of str*) – list of paths to add to the annex
- **git** (*bool*) – if True, add to git instead of annex.
- **backend** –
- **options** –
- **update** (*bool*) –

  –update option for git-add. From git's manpage:
  Update the index just where it already has an entry matching <pathspec>. This removes as well as modifies index entries to match the working tree, but adds no new files.

  If no <pathspec> is given when –update option is used, all tracked files in the entire working tree are updated (old versions of Git used to limit the update to the current directory and its subdirectories).

  Note: Used only, if a call to git-add instead of git-annex-add is performed

Return type
list of dict or dict

add_(*files*, *git=None*, *backend=None*, *options=None*, *jobs=None*, *git_options=None*, *annex_options=None*, *update=False*)

Like *add*, but returns a generator

add_url_to_file(*file_*, *url*, *options=None*, *backend=None*, *batch=False*, *git_options=None*, *annex_options=None*, *unlink_existing=False*)

Add file from url to the annex.

Downloads *file* from *url* and add it to the annex. If annex knows *file* already, records that it can be downloaded from *url*.

Note: Consider using the higher-level *download_url* instead.

Parameters

- **file** (*str*) –
- **url** (*str*) –
- **options** (*list*) – options to the annex command
- **batch** (*bool, optional*) – initiate or continue with a batched run of annex addurl, instead of just calling a single git annex addurl command
- **unlink_existing** (*bool, optional*) – by default crashes if file already exists and is under git. With this flag set to True would first remove it.

Returns
In batch mode only ATM returns dict representation of json output returned by annex

Return type
dict

add_urls(*urls*, *options=None*, *backend=None*, *cwd=None*, *jobs=None*, *git_options=None*, *annex_options=None*)

Downloads each url to its own file, which is added to the annex.

Deprecated since version 0.17: Use add_url_to_file() or call_annex() instead.

Parameters

- **urls** (*list of str*) –

- **options** (*list, optional*) – options to the annex command
- **cwd** (*string, optional*) – working directory from within which to invoke git-annex

adjust(*options=None*)

enter an adjusted branch

This command is only available in a v6+ git-annex repository.

Parameters
options (*list of str*) – currently requires '–unlock' or '–fix'; default: –unlock

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

Deprecated since version 0.16: Use get_content_annexinfo() or the test helper `datalad.tests.utils_pytest.get_annexstatus()` instead.

call_annex(*args*, *files=None*)

Call annex and return standard output.

Parameters

- **args** (*list of str*) – Arguments to pass to *annex*.
- **files** (*list of str, optional*) – File arguments to pass to *annex*. The advantage of passing these here rather than as part of *args* is that the call will be split into multiple calls to avoid exceeding the maximum command line length.

Return type
standard output (str)

Raises
See _call_annex() for information on Exceptions. –

call_annex_items_(*args*, *files=None*, *sep=None*)

Call git-annex, splitting output on *sep*.

Parameters

- **args** (*list of str*) – Arguments to pass to *git-annex*.
- **files** (*list of str, optional*) – File arguments to pass to *annex*. The advantage of passing these here rather than as part of *args* is that the call will be split into multiple calls to avoid exceeding the maximum command line length.
- **sep** (*str, optional*) – Split the output by *str.split(sep)* rather than *str.splitlines*.

Return type
Generator that yields output items.

Raises
See _call_annex() for information on Exceptions. –

call_annex_oneline(*args*, *files=None*)

Call annex for a single line of output.

This method filters prior output line selection to exclude git-annex status output that is triggered by command execution, but is not related to the particular command. This includes lines like:

(merging … into git-annex) (recording state …)

Parameters

- **args** (*list of str*) – Arguments to pass to *annex*.

- **files** (*list of str, optional*) – File arguments to pass to *annex*. The advantage of passing these here rather than as part of *args* is that the call will be split into multiple calls to avoid exceeding the maximum command line length.

Returns
Either a single output line, or an empty string if there was no output.

Return type
str

Raises

- **AssertionError if there is more than one line of output.** –
- **See _call_annex() for information on Exceptions.** –

call_annex_records(*args*, *files=None*)

Call annex with *–json** to request structured result records

This method behaves like *call_annex()*, but returns parsed result records.

Parameters

- **args** (*list of str*) – Arguments to pass to *annex*.
- **files** (*list of str, optional*) – File arguments to pass to *annex*. The advantage of passing these here rather than as part of *args* is that the call will be split into multiple calls to avoid exceeding the maximum command line length.

Returns
List of parsed result records.

Return type
list(dict)

Raises

- **CommandError if the call exits with a non-zero status. All result** –
- **records captured until the non-zero exit are available in the** –
- **exception's kwargs-dict attribute under key 'stdout_json'.** –
- **See _call_annex() for more information on Exceptions.** –

call_annex_success(*args*, *files=None*)

Call git-annex and return true if the call exit code of 0.

All parameters match those described for *call_annex*.

Return type
bool

classmethod check_direct_mode_support()

Does git-annex version support direct mode?

The result is cached at *cls.supports_direct_mode*.

Return type
bool

classmethod check_repository_versions()

Get information on supported and upgradable repository versions.

The result is cached at *cls.repository_versions*.

Returns
supported -> list of supported versions (int) upgradable -> list of upgradable versions (int)

Return type
dict

`copy_to`(*files*, *remote*, *options=None*, *jobs=None*)

Copy the actual content of *files* to *remote*

Parameters

- **files** (*str or list of str*) – path(s) to copy
- **remote** (*str*) – name of remote to copy *files* to

Returns
files successfully copied

Return type
list of str

`property default_backends`

`drop`(*files*, *options=None*, *key=False*, *jobs=None*)

Drops the content of annexed files from this repository.

Drops only if possible with respect to required minimal number of available copies.

Parameters

- **files** (*list of str*) – paths to drop
- **options** (*list of str, optional*) – commandline options for the git annex drop command
- **jobs** (*int, optional*) – how many jobs to run in parallel (passed to git-annex call)

Returns
'success' item in each object indicates failure/success per file path.

Return type
list(JSON objects)

`drop_key`(*keys*, *options=None*, *batch=False*)

Drops the content of annexed files from this repository referenced by keys

Dangerous: it drops without checking for required minimal number of available copies.

Parameters

- **keys** (*list of str, str*) –
- **batch** (*bool, optional*) – initiate or continue with a batched run of annex dropkey, instead of just calling a single git annex dropkey command

`enable_remote`(*name*, *options=None*, *env=None*)

Enables use of an existing special remote

Parameters

- **name** (*str*) – name, the special remote was created with
- **options** (*list, optional*) –

file_has_content(*files*, *allow_quick=False*, *batch=False*)

Check whether files have their content present under annex.

Parameters

- **files** (*list of str*) – file(s) to check for being actually present.
- **allow_quick** (*bool, optional*) – This is no longer supported.

Returns
For each input file states whether file has content locally

Return type
list of bool

find(*files*, *batch=False*)

Run *git annex find* on file(s).

Parameters

- **files** (*list of str*) – files to find under annex
- **batch** (*bool, optional*) – initiate or continue with a batched run of annex find, instead of just calling a single git annex find command. If any items in *files* are directories, this value is treated as False.

Returns

- A dictionary the maps each item in *files* to its *git annex find*
- *result. Items without a successful result will be an empty string, and*
- multi-item results (which can occur for if *files* includes a
- *directory) will be returned as a list.*

fsck(*paths=None*, *remote=None*, *fast=False*, *annex_options=None*, *git_options=None*)

Front-end for git-annex fsck

Parameters

- **paths** (*list*) – Limit operation to specific paths.
- **remote** (*str*) – If given, the identified remote will be fsck'ed instead of the local repository.
- **fast** (*bool*) – If True, typically means that no actual content is being verified, but tests are limited to the presence of files.

get(*files*, *remote=None*, *options=None*, *jobs=None*, *key=False*)

Get the actual content of files

Parameters

- **files** (*list of str*) – paths to get
- **remote** (*str, optional*) – from which remote to fetch content
- **options** (*list of str, optional*) – commandline options for the git annex get command
- **jobs** (*int or None, optional*) – how many jobs to run in parallel (passed to git-annex call). If not specified (None), then
- **key** (*bool, optional*) – If provided file value is actually a key

Returns
files

Return type
list of dict

`get_annexed_files`(*with_content_only=False*, *patterns=None*)

Get a list of files in annex

Parameters

- **with_content_only** (*bool, optional*) – Only list files whose content is present.
- **patterns** (*list, optional*) – Globs to pass to annex's *–include=*. Files that match any of these will be returned (i.e., they'll be separated by *–or*).

Return type
A list of POSIX file names

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

Parameters

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

Returns

The keys/values match those reported by GitRepo.get_content_info(). In addition, the following properties are added to each value dictionary:

type
Can be 'file', 'symlink', 'dataset', 'directory', where 'file' is also used for annex'ed files (corrects a 'symlink' report made by *get_content_info()*.

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

bytesize
Size of an annexed file in bytes.

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

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

Return type
dict

get_contentlocation(*key*, *batch=False*)

Get location of the key content

Normally under .git/annex objects in indirect mode and within file tree in direct mode.

Unfortunately there is no (easy) way to discriminate situations when given key is simply incorrect (not known to annex) or its content not currently present – in both cases annex just silently exits with -1

Parameters

- **key** (*str*) – key
- **batch** (*bool, optional*) – initiate or continue with a batched run of annex contentlocation

Returns

path relative to the top directory of the repository. If no content is present, empty string is returned

Return type

str

get_corresponding_branch(*branch=None*)

Get the name of a potential corresponding branch.

Parameters

branch (*str, optional*) – Name of the branch to report a corresponding branch for; defaults to active branch

Returns

Name of the corresponding branch, or *None* if there is no corresponding branch.

Return type

str or None

get_description(*uuid=None*)

Get annex repository description

Parameters

uuid (*str, optional*) – For which remote (based on uuid) to report description for

Returns

None returned if not found

Return type

str or None

get_file_annexinfo(*path*, *ref=None*, *eval_availability=False*, *key_prefix=''*)

Query annex properties for a single file

This is the companion to get_content_annexinfo() and offers simplified usage for single-file queries (the result lookup based on a path is not necessary.

All keyword arguments have identical names and semantics as their get_content_annexinfo() counterparts. See their documentation for more information.

Parameters

path (*Path or str*) – A single path to a file in the repository.

Returns

Keys and values match the values returned by get_content_annexinfo(). If a file has no annex properties (i.e., a file that is directly checked into Git and is not annexed), the returned dictionary is empty.

Return type
dict

Raises

- **ValueError** – When a given path is not matching a single file, but resolves to multiple files (e.g. a directory path)
- **NoSuchPathError** – When the given path does not match any file in a repository

get_file_backend(*files*)

Get the backend currently used for file(s).

Parameters
files (*list of str*) –

Returns
For each file in input list indicates the used backend by a str like "SHA256E" or "MD5".

Return type
list of str

get_file_key(*files*, *batch=None*)

DEPRECATED. Use get_content_annexinfo()

See the method body for how to use get_content_annexinfo() to replace get_file_key().

For single-file queries it is recommended to consider get_file_annexinfo()

get_file_size(*path*)

get_groupwanted(*name*)

Get *groupwanted* expression for a group *name*

Parameters
name (*str*) – Name of the groupwanted group

classmethod get_key_backend(*key*)

Get the backend from a given key

get_metadata(*files*, *timestamps=False*, *batch=False*)

Query git-annex file metadata

Parameters

- **files** (*str or iterable(str)*) – One or more paths for which metadata is to be queried. If one or more paths could be directories, *batch=False* must be given to prevent git-annex given an error. Due to technical limitations, such error will lead to a hanging process.
- **timestamps** (*bool, optional*) – If True, the output contains a '<metadatakey>-lastchanged' key for every metadata item, reflecting the modification time, as well as a 'lastchanged' key with the most recent modification time of any metadata item.
- **batch** (*bool, optional*) – If True, a *metadata –batch* process will be used, and only confirmed annex'ed files can be queried (else query will hang indefinitely). If False, invokes without –batch, and gives all files as arguments (this can be problematic with a large number of files).

Returns
One tuple per file (could be more items than input arguments when directories are given). First tuple item is the filename, second item is a dictionary with metadata key/value pairs.

Note that annex metadata tags are stored under the key 'tag', which is a regular metadata item that can be manipulated like any other.

Return type
generator

`get_preferred_content`(*property*, *remote=None*)

Get preferred content configuration of a repository or remote

Parameters

- **property** (*{'wanted', 'required', 'group'}*) – Type of property to query
- **remote** (*str, optional*) – If not specified (None), returns the property for the local repository.

Returns
Whether the setting is returned, or *None* if there is none.

Return type
str

Raises

- **ValueError** – If an unknown property label is given.
- **CommandError** – If the annex call errors.

`get_remotes`(*with_urls_only=False*, *exclude_special_remotes=False*)

Get known (special-) remotes of the repository

Parameters

- **exclude_special_remotes** (*bool, optional*) – if True, don't return annex special remotes
- **with_urls_only** (*bool, optional*) – return only remotes which have urls

Returns
remotes – List of names of the remotes

Return type
list of str

`static get_size_from_key`(*key*)

A little helper to obtain size encoded in a key

Returns
size of the file or None if either no size is encoded in the key or key was None itself

Return type
int or None

Raises
ValueError – if key is considered invalid (at least its size-related part)

`get_special_remotes`(*include_dead=False*)

Get info about all known (not just enabled) special remotes.

The present implementation is not able to report on special remotes that have only been configured in a private annex repo (annex.private=true).

Parameters
include_dead (*bool, optional*) – Whether to include remotes announced dead.

Returns

Keys are special remote UUIDs. Each value is a dictionary with configuration information git-annex has for the remote. This should include the 'type' and 'name' as well as any *initremote* parameters that git-annex stores.

Note: This is a faithful translation of git-annex:remote.log with one exception. For a special remote initialized with the –sameas flag, git-annex stores the special remote name under the "sameas-name" key, we copy this value under the "name" key so that callers don't have to check two places for the name. If you need to detect whether you're working with a sameas remote, the presence of either "sameas-name" or "sameas-uuid" is a reliable indicator.

Return type
dict

`get_tracking_branch`(*branch=None*, *remote_only=False*, *corresponding=True*)

Get the tracking branch for *branch* if there is any.

By default returns the tracking branch of the corresponding branch if *branch* is a managed branch.

Parameters

- **branch** (*str*) – local branch to look up. If none is given, active branch is used.
- **remote_only** (*bool*) – Don't return a value if the upstream remote is set to "." (meaning this repository).
- **corresponding** (*bool*) – If True actually look up the corresponding branch of *branch* (also if *branch* isn't explicitly given)

Returns
(remote or None, refspec or None) of the tracking branch

Return type
tuple

`get_urls`(*file_*, *key=False*, *batch=False*)

Get URLs for a file/key

Parameters

- **file** (*str*) –
- **key** (*bool, optional*) – Whether provided files are actually annex keys

Return type
A list of URLs

`git_annex_version = None`

`info`(*files*, *batch=False*, *fast=False*)

Provide annex info for file(s).

Parameters
files (*list of str*) – files to look for

Returns
Info for each file

Return type
dict

init_remote(*name*, *options*)

Creates a new special remote

Parameters
name (*str*) – name of the special remote

is_available(*file_*, *remote=None*, *key=False*, *batch=False*)

Check if file or key is available (from a remote)

In case if key or remote is misspecified, it wouldn't fail but just keep returning False, although possibly also complaining out loud ;)

Parameters

- **file** (*str*) – Filename or a key
- **remote** (*str, optional*) – Remote which to check. If None, possibly multiple remotes are checked before positive result is reported
- **key** (*bool, optional*) – Whether provided files are actually annex keys
- **batch** (*bool, optional*) – Initiate or continue with a batched run of annex checkpresentkey

Returns
with True indicating that file/key is available from (the) remote

Return type
bool

is_crippled_fs()

Return True if git-annex considers current filesystem 'crippled'.

Return type
True if on crippled filesystem, False otherwise

is_direct_mode()

Return True if annex is in direct mode

Return type
True if in direct mode, False otherwise.

is_initialized()

quick check whether this appears to be an annex-init'ed repo

is_managed_branch(*branch=None*)

Whether *branch* is managed by git-annex.

ATM this returns True if on an adjusted branch of annex v6+ repository: either 'adjusted/my_branch(unlocked)' or 'adjusted/my_branch(fixed)'

Note: The term 'managed branch' is used to make clear it's meant to be more general than the v6+ 'adjusted branch'.

Parameters
branch (*str*) – name of the branch; default: active branch

Returns
True if on a managed branch, False otherwise

Return type
bool

is_remote_annex_ignored(*remote*)

Return True if remote is explicitly ignored

is_special_annex_remote(*remote*, *check_if_known=True*)

Return whether remote is a special annex remote

Decides based on the presence of an annex- option and lack of a configured URL for the remote.

is_under_annex(*files*, *allow_quick=False*, *batch=False*)

Check whether files are under annex control

Parameters

- **files** (*list of str*) – file(s) to check for being under annex
- **allow_quick** (*bool, optional*) – This is no longer supported.

Returns
For each input file states whether file is under annex

Return type
list of bool

is_valid_annex(*allow_noninitialized=False*, *check_git=True*)

Returns whether the underlying repository appears to be still valid

Note, that this almost identical to the classmethod is_valid_repo(). However, if we are testing an existing instance, we can save Path object creations. Since this testing is done a lot, this is relevant. Creation of the Path objects in is_valid_repo() takes nearly half the time of the entire function.

Also note, that this method is bound to an instance but still class-dependent, meaning that a subclass cannot simply overwrite it. This is particularly important for the call from within __init__(), which in turn is called by the subclasses' __init__. Using an overwrite would lead to the wrong thing being called.

classmethod is_valid_repo(*path*, *allow_noninitialized=False*)

Return True if given path points to an annex repository

localsync(*remote=None*, *managed_only=False*)

Consolidate the local git-annex branch and/or managed branches.

This method calls *git annex sync* to perform purely local operations that:

1. Update the corresponding branch of any managed branch.
2. Synchronize the local 'git-annex' branch with respect to particular or all remotes (as currently reflected in the local state of their remote 'git-annex' branches).

If a repository has git-annex's 'synced/…' branches these will be updated. Otherwise, such branches that are created by *git annex sync* are removed again after the sync is complete.

Parameters

- **remote** (*str or list, optional*) – If given, specifies the name of one or more remotes to sync against. If not given, all remotes are considered.
- **managed_only** (*bool, optional*) – Only perform a sync if a managed branch with a corresponding branch is detected. By default, a sync is always performed.

merge_annex(*remote=None*)

migrate_backend(*files*, *backend=None*)

Changes the backend used for *file*.

The backend used for the key-value of *files*. Only files currently present are migrated. Note: There will be no notification if migrating fails due to the absence of a file's content!

Parameters

- **files** (*list*) – files to migrate.
- **backend** (*str*) – specify the backend to migrate to. If none is given, the default backend of this instance will be used.

precommit()

Perform pre-commit maintenance tasks, such as closing all batched annexes since they might still need to flush their changes into index

repo_info(*fast=False*, *merge_annex_branches=True*)

Provide annex info for the entire repository.

Parameters

- **fast** (*bool, optional*) – Pass *–fast* to *git annex info*.
- **merge_annex_branches** (*bool, optional*) – Whether to allow git-annex if needed to merge annex branches, e.g. to make sure up to date descriptions for git annex remotes

Returns
Info for the repository, with keys matching the ones returned by annex

Return type
dict

repository_versions = None

rm_url(*file_*, *url*)

Record that the file is no longer available at the url.

Parameters

- **file** (*str*) –
- **url** (*str*) –

set_default_backend(*backend*, *persistent=True*, *commit=True*)

Set default backend

Parameters

- **backend** (*str*) –
- **persistent** (*bool, optional*) – If persistent, would add/commit to .gitattributes. If not – would set within .git/config

set_groupwanted(*name*, *expr*)

Set *expr* for the *name* groupwanted

set_metadata(*files*, *reset=None*, *add=None*, *init=None*, *remove=None*, *purge=None*, *recursive=False*)

Manipulate git-annex file-metadata

Parameters

- **files** (*str or list(str)*) – One or more paths for which metadata is to be manipulated. The changes applied to each file item are uniform. However, the result may not be uniform across files, depending on the actual operation.
- **reset** (*dict, optional*) – Metadata items matching keys in the given dict are (re)set to the respective values.
- **add** (*dict, optional*) – The values of matching keys in the given dict appended to any possibly existing values. The metadata keys need not necessarily exist before.
- **init** (*dict, optional*) – Metadata items for the keys in the given dict are set to the respective values, if the key is not yet present in a file's metadata.
- **remove** (*dict, optional*) – Values in the given dict are removed from the metadata items matching the respective key, if they exist in a file's metadata. Non-existing values, or keys do not lead to failure.
- **purge** (*list, optional*) – Any metadata item with a key matching an entry in the given list is removed from the metadata.
- **recursive** (*bool, optional*) – If False, fail (with CommandError) when directory paths are given as *files*.

Returns
JSON obj per modified file

Return type
list

set_metadata_(*files*, *reset=None*, *add=None*, *init=None*, *remove=None*, *purge=None*, *recursive=False*)

Like set_metadata() but returns a generator

set_preferred_content(*property*, *expr*, *remote=None*)

Set preferred content configuration of a repository or remote

Parameters

- **property** (*{'wanted', 'required', 'group'}*) – Type of property to query
- **expr** (*str*) – Any expression or label supported by git-annex for the given property.
- **remote** (*str, optional*) – If not specified (None), sets the property for the local repository.

Returns
Raw git-annex output in response to the set command.

Return type
str

Raises

- **ValueError** – If an unknown property label is given.
- **CommandError** – If the annex call errors.

set_remote_dead(*name*)

Announce to annex that remote is "dead"

set_remote_url(*name*, *url*, *push=False*)

Set the URL a remote is pointing to

Sets the URL of the remote *name*. Requires the remote to already exist.

Parameters

- **name** (*str*) – name of the remote
- **url** (*str*) –
- **push** (*bool*) – if True, set the push URL, otherwise the fetch URL; if True, additionally set annexurl to *url*, to make sure annex uses it to talk to the remote, since access via fetch URL might be restricted.

supports_direct_mode = None

property supports_unlocked_pointers

Return True if repository version supports unlocked pointers.

sync(*remotes=None*, *push=True*, *pull=True*, *commit=True*, *content=False*, *all=False*, *fast=False*)

This method is deprecated, use call_annex(['sync', ...]) instead.

Synchronize local repository with remotes

Use this command when you want to synchronize the local repository with one or more of its remotes. You can specify the remotes (or remote groups) to sync with by name; the default if none are specified is to sync with all remotes.

Parameters

- **remotes** (*str, list(str), optional*) – Name of one or more remotes to be sync'ed.
- **push** (*bool*) – By default, git pushes to remotes.
- **pull** (*bool*) – By default, git pulls from remotes
- **commit** (*bool*) – A commit is done by default. Disable to avoid committing local changes.
- **content** (*bool*) – Normally, syncing does not transfer the contents of annexed files. This option causes the content of files in the work tree to also be uploaded and downloaded as necessary.
- **all** (*bool*) – This option, when combined with *content*, makes all available versions of all files be synced, when preferred content settings allow
- **fast** (*bool*) – Only sync with the remotes with the lowest annex-cost value configured

unannex(*files*, *options=None*)

undo accidental add command

Use this to undo an accidental git annex add command. Note that for safety, the content of the file remains in the annex, until you use git annex unused and git annex dropunused.

Parameters

- **files** (*list of str*) –
- **options** (*list of str*) –

Returns
successfully unannexed files

Return type
list of str

unlock(*files*)

unlock files for modification

Note: This method is silent about errors in unlocking a file (e.g, the file has not content). Use the higher-level interface.unlock to get more informative reporting.

Parameters

files (*list of str*) –

Returns

successfully unlocked files

Return type

list of str

property uuid

Annex UUID

Returns

Returns a the annex UUID, if there is any, or *None* otherwise.

Return type

str

whereis(*files*, *output='uuids'*, *key=False*, *options=None*, *batch=False*)

Lists repositories that have actual content of file(s).

Parameters

- **files** (*list of str*) – files to look for
- **output** (*{'descriptions', 'uuids', 'full'}, optional*) – If 'descriptions', a list of remotes descriptions returned is per each file. If 'full', for each file a dictionary of all fields is returned as returned by annex
- **key** (*bool, optional*) – Whether provided files are actually annex keys
- **options** (*list, optional*) – Options to pass into git-annex call

Returns

if output == 'descriptions', contains a list of descriptions of remotes for each input file, describing the remote for each remote, which was found by git-annex whereis, like:

```
u'me@mycomputer:~/where/my/repo/is [origin]' or
u'web' or
u'me@mycomputer:~/some/other/clone'
```

if output == 'uuids', returns a list of uuids. if output == 'full', returns a dictionary with filenames as keys and values a detailed record, e.g.:

```
{'00000000-0000-0000-0000-000000000001': {
  'description': 'web',
  'here': False,
  'urls': ['http://127.0.0.1:43442/about.txt', 'http://example.com/
↪someurl']
}}
```

Return type

list of list of unicode or dict

class datalad.support.annexrepo.**BatchedAnnex**(*annex_cmd*, *git_options=None*, *annex_options=None*, *path=None*, *json=False*, *output_proc=None*, *batch_opt='--batch'*)

Bases: *BatchedCommand*

Container for an annex process which would allow for persistent communication

class datalad.support.annexrepo.**BatchedAnnexes**(*batch_size=0*, *git_options=None*)

Bases: *SafeDelCloseMixin*, dict

Class to contain the registry of active batch'ed instances of annex for a repository

clear()

Override just to make sure we don't rely on __del__ to close all the pipes

close()

Close communication to all the batched annexes

It does not remove them from the dictionary though

get(*codename*, *annex_cmd=None*, ***kwargs*)

Return the value for key if key is in the dictionary, else default.

Return type
BatchedAnnex

class datalad.support.annexrepo.**GeneratorAnnexJsonNoStderrProtocol**(*done_future=None*, *total_nbytes=None*)

Bases: *GeneratorAnnexJsonProtocol*

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

pipe_data_received(*fd*, *data*)

process: Optional[subprocess.Popen]

process_exited()

class datalad.support.annexrepo.**GeneratorAnnexJsonProtocol**(*done_future=None*, *total_nbytes=None*)

Bases: *GeneratorMixIn*, *AnnexJsonProtocol*

add_to_output(*json_object*)

fd_infos: dict[int, tuple[str, Optional[bytearray]]]

process: Optional[subprocess.Popen]

datalad.support.annexrepo.**readline_json**(*stdout*)

datalad.support.annexrepo.**readlines_until_ok_or_failed**(*stdout*, *maxlines=100*)

Read stdout until line ends with ok or failed

datalad.support.archives

Various handlers/functionality for different types of files (e.g. for archives)

class datalad.support.archives.**ArchivesCache**(*toppath=None*, *persistent=False*)

Bases: object

Cache to maintain extracted archives

Parameters

- **toppath** (*str*) – Top directory under .git/ of which temp directory would be created. If not provided – random tempdir is used
- **persistent** (*bool, optional*) – Passed over into generated ExtractedArchives

clean(*force=False*)

get_archive(*archive*)

property path

class datalad.support.archives.**ExtractedArchive**(*archive*, *path=None*, *persistent=False*)

Bases: object

Container for the extracted archive

STAMP_SUFFIX = '.stamp'

assure_extracted()

Return path to the extracted *archive*. Extract archive if necessary

clean(*force=False*)

get_extracted_file(*afile*)

get_extracted_filename(*afile*)

Return full path to the *afile* within extracted *archive*

It does not actually extract any archive

get_extracted_files()

Generator to provide filenames which are available under extracted archive

get_leading_directory(*depth=None*, *consider=None*, *exclude=None*)

Return leading directory of the content within archive

Parameters

- **depth** (*int or None, optional*) – Maximal depth of leading directories to consider. If None - no upper limit
- **consider** (*list of str, optional*) – Regular expressions for file/directory names to be considered (before exclude). Applied to the entire relative path to the file as in the archive
- **exclude** (*list of str, optional*) – Regular expressions for file/directory names to be excluded from consideration. Applied to the entire relative path to the file as in the archive

Returns

If there is no single leading directory – None returned

Return type
str or None

property is_extracted

property path

Given an archive – return full path to it within cache (extracted)

property stamp_path

datalad.support.archives.**decompress_file**(*archive*, *dir_*, *leading_directories='strip'*)

Decompress *archive* into a directory *dir_*

Parameters

- **archive** (*str*) –
- **dir** (*str*) –
- **leading_directories** (*{'strip', None}*) – If *strip*, and archive contains a single leading directory under which all content is stored, all the content will be moved one directory up and that leading directory will be removed.

datalad.support.extensions

Support functionality for extension development

datalad.support.extensions.**has_config**(*name*)

Returns whether a configuration item is registered under the given name

Parameters
name (*str*) – Configuration item name

Return type
bool

datalad.support.extensions.**register_config**(*name*, *title*, ***, *default=<class 'datalad.interface.common_cfg._NotGiven'>*, *default_fn=<class 'datalad.interface.common_cfg._NotGiven'>*, *description=None*, *type=<class 'datalad.interface.common_cfg._NotGiven'>*, *dialog=None*, *scope=<class 'datalad.interface.common_cfg._NotGiven'>*)

Register a configuration item

This function can be used by DataLad extensions and other client code to register configurations items and their documentation with DataLad's configuration management. Specifically, these definitions will be interpreted by and acted on by the *configuration* command, and *ConfigManager.obtain()*.

At minimum, each item must be given a name, and a title. Optionally, any configuration item can be given a default (or a callable to compute a default lazily on access), a type-defining/validating callable (i.e. *Constraint*), a (longer) description, a dialog type to enable manual entry, and a configuration scope to store entered values in.

Parameters

- **name** (*str*) – Configuration item name, in most cases starting with the prefix 'datalad.' followed by at least a section name, and a variable name, e.g. 'datalad.section.variable', following Git's syntax for configuration items.

- **title** (*str*) – The briefest summary of the configuration item's purpose, typically written in the style of a headline for a dialog UI, or that of an explanatory inline comment just prior the item definitions.
- **default** (*optional*) – A default value that is already known at the time of registering the configuration items. Can be of any type.
- **default_fn** (*callable, optional*) – A callable to compute a default value lazily on access. The can be used, if the actual value is not yet known at the time of registering the configuration item, or if the default is expensive to compute and its evaluation needs to be deferred to prevent slow startup (configuration items are typically defined as one of the first things on import).
- **description** (*str, optional*) – A longer description to accompany the title, possibly with instructions on how a sensible value can be determined, or with details on the impact of a configuration switch.
- **type** (*callable, optional*) – A callable to perform arbitrary type conversion and validation of value (or default values). If validation/conversion fails, the callable must raise an arbitrary exception. The *str(callable)* is used as a type description.
- **dialog** (*{'yesno', 'question'}*) – A type of UI dialog to use when manual value entry is attempted (only in interactive sessions, and only when no default is defined. *title* and *description* will be displayed in this dialog.
- **scope** (*{'override', 'global', 'local', 'branch'}, optional*) – If particular code requests the storage of (manually entered) values, but defines no configuration scope, this default scope will be used.

Raises
ValueError – For missing required, or invalid configuration properties.

datalad.customremotes.base

Base classes to custom git-annex remotes (e.g. extraction from archives)

class datalad.customremotes.base.**AnnexCustomRemote**(*annex*)

Bases: SpecialRemote

AVAILABILITY = 'local'

COST = 100

gen_URLS(*key*)

Yield URL(s) associated with a key, and keep stats on protocols.

getavailability()

Asks the remote if it is locally or globally available. (Ie stored in the cloud vs on a local disk.)

Returns
Allowed values are "global" or "local".

Return type
str

getcost()

Requests the remote to return a use cost. Higher costs are more expensive.

cheapRemoteCost = 100 nearlyCheapRemoteCost = 110 semiExpensiveRemoteCost = 175 expensiveRemoteCost = 200 veryExpensiveRemoteCost = 1000 (taken from Config/Cost.hs)

Returns
Indicates the cost of the remote.

Return type
int

`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.

`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_store(key, local_file)`

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.

`datalad.customremotes.base.ensure_datalad_remote(repo, remote=None, encryption=None, autoenable=False)`

Initialize and enable datalad special remote if it isn't already.

Parameters

- `repo` (`AnnexRepo`) –

- **remote** (*str, optional*) – Special remote name. This should be one of the values in datalad.consts.DATALAD_SPECIAL_REMOTES_UUIDS and defaults to datalad.consts.DATALAD_SPECIAL_REMOTE.
- **encryption** (*optional*) – Passed to *init_datalad_remote*.
- **autoenable** (*optional*) – Passed to *init_datalad_remote*.

datalad.customremotes.base.**generate_uuids**()

Generate UUIDs for our remotes. Even though quick, for consistency pre-generated and recorded in consts.py

datalad.customremotes.base.**init_datalad_remote**(*repo*, *remote*, *encryption=None*, *autoenable=False*, *opts=[]*)

Initialize datalad special remote

datalad.customremotes.archives

Custom remote to get the load from archives present under annex

class datalad.customremotes.archives.**ArchiveAnnexCustomRemote**(*annex*, *path=None*, *persistent_cache=True*, ***kwargs*)

Bases: *AnnexCustomRemote*

Special custom remote allowing to obtain files from archives

Archives must be under annex'ed themselves.

COST = 500

CUSTOM_REMOTE_NAME = 'archive'

SUPPORTED_SCHEMES = ('dl+archive',)

URL_PREFIX = 'dl+archive:'

URL_SCHEME = 'dl+archive'

property cache

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, eg. in case of connection error.

checkurl(*url*)

Asks the remote to check if the url's content can currently be downloaded (without downloading it). The remote can optionally provide additional information about the file.

Parameters
url (*str*) –

Returns

True if the url's content can currently be downloaded and no additional information can be provided. False if it can't currently be downloaded.

In order to provide additional information, a list of dictionaries can be returned. The dictionaries can have 3 keys: {'url': str, 'size': int, 'filename': str}. All of them are optional.

If there is only one file to be downloaded, we could return: [{'size': 512, 'filename':'example_file.txt'}]

Other examples: {'url':"https://example.com", 'size':512, 'filename':"example_file.txt"} [{'url':"Url1", 'size':512, 'filename':"Filename1"}, {'url':"Url2", 'filename':"Filename2"}]

Return type
Union(bool, List(Dict))

claimurl(*url*)

Asks the remote if it wishes to claim responsibility for downloading an url.

Parameters
url (*str*) –

Returns
True if it wants to claim this url. False if it doesn't.

Return type
bool

get_contentlocation(*key*, *absolute=False*, *verify_exists=True*)

Return (relative to top or absolute) path to the file containing the key

This is a wrapper around AnnexRepo.get_contentlocation which provides caching of the result (we are asking the location for the same archive key often)

get_file_url(*archive_file=None*, *archive_key=None*, *file=None*, *size=None*)

Given archive (file or a key) and a file – compose URL for access

Examples

dl+archive:SHA256E-s176–69…3e.tar.gz#path=1/d2/2d&size=123
when size of file within archive was known to be 123

dl+archive:SHA256E-s176–69…3e.tar.gz#path=1/d2/2d
when size of file within archive was not provided

Parameters
size (*int, optional*) – Size of the file. If not provided, will simply be empty

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.

stop(**args*)

Stop communication with annex

transfer_retrieve(*key*, *file*)

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.

whereis(*key*)

Asks the remote to provide additional information about ways to access the content of a key stored in it, such as eg, public urls. This will be displayed to the user by eg, git annex whereis. Note that users expect git annex whereis to run fast, without eg, network access.

Parameters

key (*str*) –

Returns

Information about the location of the key, eg. public urls.

Return type

str

datalad.customremotes.archives.**link_file_load**(*src*, *dst*, *dry_run=False*)

Just a little helper to hardlink files's load

datalad.customremotes.archives.**main**()

cmdline entry point

datalad.runner.nonasyncrunner

Thread based subprocess execution with stdout and stderr passed to protocol objects

class datalad.runner.nonasyncrunner.**ThreadedRunner**(*cmd*, *protocol_class*, *stdin*, *protocol_kwargs=None*, *timeout=None*, *exception_on_error=True*, ***popen_kwargs*)

Bases: object

A class the contains a naive implementation for concurrent sub-process execution. It uses *subprocess.Popen* and threads to read from stdout and stderr of the subprocess, and to write to stdin of the subprocess.

All read data and timeouts are passed to a protocol instance, which can create the final result.

Parameters

- **cmd** (str | list) –
- **protocol_class** (type[WitlessProtocol]) –
- **stdin** (int | IO | bytes | Queue[Optional[bytes]] | None) –

- **protocol_kwargs** (Optional[dict]) –
- **timeout** (Optional[float]) –
- **exception_on_error** (bool) –

check_for_stall()

Return type
bool

close_stdin()

ensure_stdin_stdout_stderr_closed()

ensure_stdout_stderr_closed()

is_stalled()

Return type
bool

process_loop()

Return type
dict

process_queue()

Get a single event from the queue or handle a timeout. This method might modify the set of active file numbers if a file-closed event is read from the output queue, or if a timeout-callback return True.

process_timeouts()

Check for timeouts

This method checks whether a timeout occurred since it was called last. If a timeout occurred, the timeout handler is called.

Return type
bool

Returns: bool
Return *True* if at least one timeout occurred, *False* if no timeout occurred.

remove_file_number(*file_number*)

Remove a file number from the active set and from the timeout set.

Parameters
file_number (int) –

remove_process()

run()

Run the command as specified in __init__.

This method is not re-entrant. Furthermore, if the protocol is a subclass of GeneratorMixIn, and the generator has not been exhausted, i.e. it has not raised *StopIteration*, this method should not be called again. If it is called again before the generator is exhausted, a RuntimeError is raised. In the non-generator case, a second caller will be suspended until the first caller has returned.

Return type
dict | _ResultGenerator

Returns

- *Any* – If the protocol is not a subclass of `GeneratorMixIn`, the result of protocol._prepare_result will be returned.
- *Generator* – If the protocol is a subclass of `GeneratorMixIn`, a Generator will be returned. This allows to use this method in constructs like:

```
for protocol_output in runner.run():
    ...
```

Where the iterator yields whatever protocol.pipe_data_received sends into the generator. If all output was yielded and the process has terminated, the generator will raise StopIteration(return_code), where return_code is the return code of the process. The return code of the process will also be stored in the "return_code"-attribute of the runner. So you could write:

```
gen = runner.run()
for file_descriptor, data in gen:
    ...

# get the return code of the process
result = gen.return_code
```

should_continue()

Return type
bool

timeout_resolution = 0.2

wait_for_threads()

datalad.runner.nonasyncrunner.**run_command**(*cmd*, *protocol*, *stdin*, *protocol_kwargs=None*, *timeout=None*, *exception_on_error=True*, ***popen_kwargs*)

Run a command in a subprocess

this function delegates the execution to an instance of *ThreadedRunner*, please see *ThreadedRunner.__init__()* for a documentation of the parameters, and *ThreadedRunner.run()* for a documentation of the return values.

Parameters

- **cmd** (str | list) –
- **protocol** (type[WitlessProtocol]) –
- **stdin** (int | IO | bytes | Queue[Optional[bytes]] | None) –
- **protocol_kwargs** (Optional[dict]) –
- **timeout** (Optional[float]) –
- **exception_on_error** (bool) –

Return type
dict | _ResultGenerator

datalad.runner.protocol

Base class of a protocol to be used with the DataLad runner

class datalad.runner.protocol.**GeneratorMixIn**

Bases: object

Protocol mix in that will instruct runner.run to return a generator

When this class is in the parent of a protocol given to runner.run (and some other functions/methods) the run-method will return a *Generator*, which yields whatever the protocol callbacks send to the *Generator*, via the *send_result*-method of this class.

This allows to use runner.run() in constructs like:

for result in runner.run(...):
do something, for example write to stdin of the subprocess

send_result(*result*)

class datalad.runner.protocol.**WitlessProtocol**(*done_future=None*, *encoding=None*)

Bases: object

Subprocess communication protocol base class for *run_async_cmd*

This class implements basic subprocess output handling. Derived classes like *StdOutCapture* should be used for subprocess communication that need to capture and return output. In particular, the *pipe_data_received()* method can be overwritten to implement "online" processing of process output.

This class defines a default return value setup that causes *run_async_cmd()* to return a 2-tuple with the subprocess's exit code and a list with bytestrings of all captured output streams.

Parameters

- **done_future** (Optional[Any]) –
- **encoding** (Optional[str]) –

connection_lost(*exc*)

Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

Parameters
exc (Optional[BaseException]) –

Return type
None

connection_made(*process*)

Parameters
process (Popen) –

Return type
None

pipe_connection_lost(*fd*, *exc*)

Called when a file descriptor associated with the child process is closed.

fd is the int file descriptor that was closed.

Parameters

- **fd** (int) –
- **exc** (Optional[BaseException]) –

Return type
None

pipe_data_received(*fd*, *data*)

Parameters

- **fd** (int) –
- **data** (bytes) –

Return type
None

proc_err = False

proc_out = False

process_exited()

Return type
None

timeout(*fd*)

Called if the timeout parameter to WitlessRunner.run() is not *None* and a process file descriptor could not be read (stdout or stderr) or not be written (stdin) within the specified time in seconds, or if waiting for a subprocess to exit takes longer than the specified time.

stdin timeouts are only caught when the type of the *stdin-* parameter to WitlessRunner.run() is either a *Queue*, a *str*, or *bytes*. *Stdout* or *stderr* timeouts are only caught of proc_out and proc_err are *True* in the protocol class. Process wait timeouts are always caught if *timeout* is not *None*. In this case the *fd*-argument will be *None*.

fd:
The file descriptor that timed out or *None* if no progress was made at all, i.e. no stdin element was enqueued and no output was read from either stdout or stderr.

Return type
bool

Returns
If the callback returns *True*, the file descriptor (if any was given) will be closed and no longer monitored. If the return values is anything else than *True*, the file-descriptor will be monitored further and additional timeouts might occur indefinitely. If *None* was given, i.e. a process runtime-timeout was detected, and *True* is returned, the process will be terminated.

Parameters
fd (Optional[int]) –

Configuration management

config

datalad.config

class datalad.config.**ConfigManager**(*dataset=None*, *overrides=None*, *source='any'*)

Bases: `object`

Thin wrapper around *git-config* with support for a dataset configuration.

The general idea is to have an object that is primarily used to read/query configuration option. Upon creation, current configuration is read via one (or max two, in the case of the presence of dataset-specific configuration) calls to *git config*. If this class is initialized with a Dataset instance, it supports reading and writing configuration from `.datalad/config` inside a dataset too. This file is committed to Git and hence useful to ship certain configuration items with a dataset.

The API aims to provide the most significant read-access API of a dictionary, the Python ConfigParser, and GitPython's config parser implementations.

This class is presently not capable of efficiently writing multiple configurations items at once. Instead, each modification results in a dedicated call to *git config*. This author thinks this is OK, as he cannot think of a situation where a large number of items need to be written during normal operation.

Each instance carries a public *overrides* attribute. This dictionary contains variables that override any setting read from a file. The overrides are persistent across reloads.

Any DATALAD_* environment variable is also presented as a configuration item. Settings read from environment variables are not stored in any of the configuration files, but are read dynamically from the environment at each *reload()* call. Their values take precedence over any specification in configuration files, and even overrides.

Parameters

- **dataset** (*Dataset, optional*) – If provided, all *git config* calls are executed in this dataset's directory. Moreover, any modifications are, by default, directed to this dataset's configuration file (which will be created on demand)
- **overrides** (*dict, optional*) – Variable overrides, see general class documentation for details.
- **source** (*{'any', 'local', 'branch', 'branch-local'}, optional*) – Which sources of configuration setting to consider. If 'branch', configuration items are only read from a dataset's persistent configuration file in current branch, if any is present (the one in `.datalad/config`, not `.git/config`); if 'local', any non-committed source is considered (local and global configuration in Git config's terminology); if 'branch-local', persistent configuration in current dataset branch and local, but not global or system configuration are considered; if 'any' all possible sources of configuration are considered. Note: 'dataset' and 'dataset-local' are deprecated in favor of 'branch' and 'branch-local'.

add(*var*, *value*, *scope='branch'*, *reload=True*)

Add a configuration variable and value

Parameters

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. 'core.editor'

- **value** (*str*) – Variable value
- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

Parameters

- **default** (*optional*) – Value to return when key is not present. *None* by default.
- **get_all** (*bool, optional*) – If True, return all values of multiple identical configuration keys. By default only the last specified value is returned.

get_from_source(*source*, *key*, *default=None*)

Like get(), but a source can be specific.

If *source* is 'branch', only the committed configuration is queried, overrides are applied. In the case of 'local', the committed configuration is ignored, but overrides and configuration from environment variables are applied as usual.

get_value(*section*, *option*, *default=None*)

Like *get()*, but with an optional default value

If the default is not None, the given default value will be returned in case the option did not exist. This behavior imitates GitPython's config parser.

getbool(*section*, *option*, *default=None*)

A convenience method which coerces the option value to a bool

Values "on", "yes", "true" and any int!=0 are considered True Values which evaluate to bool False, "off", "no", "false" are considered False TypeError is raised for other values.

getfloat(*section*, *option*)

A convenience method which coerces the option value to a float

getint(*section*, *option*)

A convenience method which coerces the option value to an integer

has_option(*section*, *option*)

If the given section exists, and contains the given option

has_section(*section*)

Indicates whether a section is present in the configuration

items(*section=None*)

Return a list of (name, value) pairs for each option

Optionally limited to a given section.

keys()

Returns list of configuration item names

obtain(*var*, *default=None*, *dialog_type=None*, *valtype=None*, *store=False*, *scope=None*, *reload=True*, ***kwargs*)

Convenience method to obtain settings interactively, if needed

A UI will be used to ask for user input in interactive sessions. Questions to ask, and additional explanations can be passed directly as arguments, or retrieved from a list of pre-configured items.

Additionally, this method allows for type conversion and storage of obtained settings. Both aspects can also be pre-configured.

Parameters

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. 'core.editor'
- **default** (*any type*) – In interactive sessions and if *store* is True, this default value will be presented to the user for confirmation (or modification). In all other cases, this value will be silently assigned unless there is an existing configuration setting.
- **dialog_type** (*{'question', 'yesno', None}*) – Which dialog type to use in interactive sessions. If *None*, pre-configured UI options are used.
- **store** (*bool*) – Whether to store the obtained value (or default)
- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.
- ****kwargs** – Additional arguments for the UI function call, such as a question *text*.

options(*section*)

Returns a list of options available in the specified section.

reload(*force=False*)

Reload all configuration items from the configured sources

If *force* is False, all files configuration was previously read from are checked for differences in the modification times. If no difference is found for any file no reload is performed. This mechanism will not detect newly created global configuration files, use *force* in this case.

remove_section(*sec*, *scope='branch'*, *reload=True*)

Rename a configuration section

Parameters

- **sec** (*str*) – Name of the section to remove.

- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

rename_section(*old*, *new*, *scope='branch'*, *reload=True*)

Rename a configuration section

Parameters

- **old** (*str*) – Name of the section to rename.
- **new** (*str*) – Name of the section to rename to.
- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

rewrite_url(*url*)

Any matching 'url.<base>.insteadOf' configuration is applied

Any URL that starts with such a configuration will be rewritten to start, instead, with <base>. When more than one insteadOf strings match a given URL, the longest match is used.

Parameters

- **cfg** (*ConfigManager or dict*) – dict-like with configuration variable name/value-pairs.
- **url** (*str*) – URL to be rewritten, if matching configuration is found.

Returns

Rewritten or unmodified URL.

Return type

str

sections()

Returns a list of the sections available

set(*var*, *value*, *scope='branch'*, *reload=True*, *force=False*)

Set a variable to a value.

In opposition to *add*, this replaces the value of *var* if there is one already.

Parameters

- **var** (*str*) – Variable name including any section like *git config* expects them, e.g. 'core.editor'
- **value** (*str*) – Variable value
- **force** (*bool*) – if set, replaces all occurrences of *var* by a single one with the given *value*. Otherwise raise if multiple entries for *var* exist already
- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

unset(*var*, *scope='branch'*, *reload=True*)

Remove all occurrences of a variable

Parameters

- **var** (*str*) – Name of the variable to remove
- **scope** (*{'branch', 'local', 'global', 'override'}, optional*) – Indicator which configuration file to modify. 'branch' indicates the persistent configuration in .datalad/config of a dataset; 'local' the configuration of a dataset's Git repository in .git/config; 'global' refers to the general configuration that is not specific to a single repository (usually in $USER/.gitconfig); 'override' limits the modification to the ConfigManager instance, and the assigned value overrides any setting from any other source. Note: 'dataset' is being DEPRECATED in favor of 'branch'.
- **where** (*{'branch', 'local', 'global', 'override'}, optional*) – DEPRECATED, use 'scope'.
- **reload** (*bool*) – Flag whether to reload the configuration from file(s) after modification. This can be disable to make multiple sequential modifications slightly more efficient.

datalad.config.**anything2bool**(*val*)

datalad.config.**get_git_version**(*runner=None*)

Return version of available git

datalad.config.**parse_gitconfig_dump**(*dump*, *cwd=None*, *multi_value=True*)

Parse a dump-string from *git config -z –list*

This parser has limited support for discarding unrelated output that may contaminate the given dump. It does so performing a relatively strict matching of configuration key syntax, and discarding lines in the output that are not valid git-config keys.

There is also built-in support for parsing outputs generated with –show-origin (see return value).

Parameters

- **dump** (*str*) – Null-byte separated output

- **cwd** (*path-like, optional*) – Use this absolute path to convert relative paths for origin reports into absolute paths. By default, the process working directory PWD is used.
- **multi_value** (*bool, optional*) – If True, report values from multiple specifications of the same key as a tuple of values assigned to this key. Otherwise, the last configuration is reported.
- **Returns** –
- **--------** –
- **dict** – Configuration items are returned as key/value pairs in a dictionary. The second tuple-item will be a set of identifiers comprising all source files/blobs, if origin information was included in the dump (–show-origin). An empty set is returned otherwise. For actual files a Path object is included in the set, for a git-blob a Git blob ID prefixed with 'blob:' is reported.
- **set** – Configuration items are returned as key/value pairs in a dictionary. The second tuple-item will be a set of identifiers comprising all source files/blobs, if origin information was included in the dump (–show-origin). An empty set is returned otherwise. For actual files a Path object is included in the set, for a git-blob a Git blob ID prefixed with 'blob:' is reported.

datalad.config.**quote_config**(*v*)

Helper to perform minimal quoting of config keys/value parts

Parameters
v (*str*) – To-be-quoted string

datalad.config.**rewrite_url**(*cfg*, *url*)

Any matching 'url.<base>.insteadOf' configuration is applied

Any URL that starts with such a configuration will be rewritten to start, instead, with <base>. When more than one insteadOf strings match a given URL, the longest match is used.

Parameters

- **cfg** (*ConfigManager or dict*) – dict-like with configuration variable name/value-pairs.
- **url** (*str*) – URL to be rewritten, if matching configuration is found.

Returns
Rewritten or unmodified URL.

Return type
str

datalad.config.**warn_on_undefined_git_identity**(*cfg*)

Check whether a Git identity is defined, and warn if not

Parameters
cfg (*ConfigManager*) –

datalad.config.**write_config_section**(*fobj*, *suite*, *name*, *props*)

Write a config section with (multiple) settings.

Parameters

- **fobj** (*File*) – Opened target file
- **suite** (*str*) – First item of the section name, e.g. 'submodule', or 'datalad'
- **name** (*str*) – Remainder of the section name
- **props** (*dict*) – Keys are configuration setting names within the section context (i.e. not duplicating *suite* and/or *name*, values are configuration setting values.

Test infrastructure

tests.utils_pytest	Miscellaneous utilities to assist with testing
tests.utils_testrepos	
tests.heavyoutput	Helper to provide heavy load on stdout and stderr

datalad.tests.utils_pytest

Miscellaneous utilities to assist with testing

class datalad.tests.utils_pytest.**HTTPPath**(*path*, *use_ssl=False*, *auth=None*)

Bases: `object`

Serve the content of a path via an HTTP URL.

This class can be used as a context manager, in which case it returns the URL.

Alternatively, the *start* and *stop* methods can be called directly.

Parameters

- **path** (*str*) – Directory with content to serve.
- **use_ssl** (*bool*) –
- **auth** (*tuple*) – Username, password

start()

Start serving *path* via HTTP.

stop()

Stop serving *path*.

class datalad.tests.utils_pytest.**SilentHTTPHandler**(**args*, ***kwargs*)

Bases: `SimpleHTTPRequestHandler`

A little adapter to silence the handler

log_message(*format*, **args*)

Log an arbitrary message.

This is used by all other logging functions. Override it if you have specific logging wishes.

The first argument, FORMAT, is a format string for the message to be logged. If the format string contains any % escapes requiring parameters, they should be specified as subsequent arguments (it's just like printf!).

The client ip and current date/time are prefixed to every message.

Unicode control characters are replaced with escaped hex before writing the output to stderr.

datalad.tests.utils_pytest.**assert_cwd_unchanged**(*func*, *ok_to_chdir=False*)

Decorator to test whether the current working directory remains unchanged

Parameters

ok_to_chdir (*bool, optional*) – If True, allow to chdir, so this decorator would not then raise exception if chdir'ed but only return to original directory

datalad.tests.utils_pytest.**assert_dict_equal**(*d1*, *d2*)

datalad.tests.utils_pytest.**assert_equal**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_false**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**assert_greater**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_greater_equal**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_in**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_in_results**(*results*, ***kwargs*)

Verify that the particular combination of keys and values is found in one of the results

datalad.tests.utils_pytest.**assert_is**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_is_generator**(*gen*)

datalad.tests.utils_pytest.**assert_is_instance**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_is_none**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**assert_is_not**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_is_not_none**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**assert_message**(*message*, *results*)

Verify that each status dict in the results has a message

This only tests the message template string, and not a formatted message with args expanded.

datalad.tests.utils_pytest.**assert_no_errors_logged**(*func*, *skip_re=None*)

Decorator around function to assert that no errors logged during its execution

datalad.tests.utils_pytest.**assert_not_equal**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_not_in**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_not_in_results**(*results*, ***kwargs*)

Verify that the particular combination of keys and values is not in any of the results

datalad.tests.utils_pytest.**assert_not_is_instance**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_re_in**(*regex*, *c*, *flags=0*, *match=True*, *msg=None*)

Assert that container (list, str, etc) contains entry matching the regex

datalad.tests.utils_pytest.**assert_repo_status**(*path*, *annex=None*, *untracked_mode='normal'*, ***kwargs*)

Compare a repo status against (optional) exceptions.

Anything file/directory that is not explicitly indicated must have state 'clean', i.e. no modifications and recorded in Git.

Parameters

- **path** (*str or Repo*) – in case of a str: path to the repository's base dir; Note, that passing a Repo instance prevents detecting annex. This might be useful in case of a non-initialized annex, a GitRepo is pointing to.
- **annex** (*bool or None*) – explicitly set to True or False to indicate, that an annex is (not) expected; set to None to autodetect, whether there is an annex. Default: None.

- **untracked_mode** (*{'no', 'normal', 'all'}*) – If and how untracked content is reported. The specification of untracked files that are OK to be found must match this mode. See *Repo.status()*
- ****kwargs** – Files/directories that are OK to not be in 'clean' state. Each argument must be one of 'added', 'untracked', 'deleted', 'modified' and each value must be a list of filenames (relative to the root of the repository, in POSIX convention).

datalad.tests.utils_pytest.**assert_result_count**(*results*, *n*, ***kwargs*)

Verify specific number of results (matching criteria, if any)

datalad.tests.utils_pytest.**assert_result_values_cond**(*results*, *prop*, *cond*)

Verify that the values of all results for a given key in the status dicts fulfill condition *cond*.

Parameters

- **results** –
- **prop** (*str*) –
- **cond** (*callable*) –

datalad.tests.utils_pytest.**assert_result_values_equal**(*results*, *prop*, *values*)

Verify that the values of all results for a given key in the status dicts match the given sequence

datalad.tests.utils_pytest.**assert_set_equal**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**assert_status**(*label*, *results*)

Verify that each status dict in the results has a given status label

label can be a sequence, in which case status must be one of the items in this sequence.

datalad.tests.utils_pytest.**assert_str_equal**(*s1*, *s2*)

Helper to compare two lines

datalad.tests.utils_pytest.**assert_true**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**attr**(*name*)

datalad.tests.utils_pytest.**check_not_generatorfunction**(*func*)

Internal helper to verify that we are not decorating generator tests

datalad.tests.utils_pytest.**eq_**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**get_annexstatus**(*ds*, *paths=None*)

Report a status for annexed contents. Assembles states for git content info, amended with annex info on 'HEAD' (to get the last committed stage and with it possibly vanished content), and lastly annex info wrt to the present worktree, to also get info on added/staged content this fuses the info reported from - git ls-files - git annex findref HEAD - git annex find –include '*'

datalad.tests.utils_pytest.**get_convoluted_situation**(*path*, *repocls=<class 'datalad.support.annexrepo.AnnexRepo'>*)

datalad.tests.utils_pytest.**get_datasets_topdir**()

Delayed parsing so it could be monkey patched etc

datalad.tests.utils_pytest.**get_deeply_nested_structure**(*path*)

Here is what this does (assuming UNIX, locked): | . | ├── directory_untracked | │ └── link2dir -> ../subdir | ├── OBSCURE_FILENAME_file_modified | ├── link2dir -> subdir | ├── link2subdsdir -> subds_modified/subdir | ├── link2subdsroot -> subds_modified | ├── subdir | │ ├── annexed_file.txt -> ../.git/annex/objects/… | │ ├── file_modified | │ ├── git_file.txt | │ └── link2annex_files.txt -> annexed_file.txt | └── subds_modified

| ├── link2superdsdir -> ../subdir | ├── subdir | | └── annexed_file.txt -> ../.git/annex/objects/… | └── subds_lvl1_modified | └── OBSCURE_FILENAME_directory_untracked | └── untracked_file

When a system has no symlink support, the link2… components are not included.

`datalad.tests.utils_pytest.`**`get_most_obscure_supported_name`**(*tdir*, *return_candidates=False*)

Return the most obscure filename that the filesystem would support under TEMPDIR

Parameters

- **return_candidates** (*bool, optional*) – if True, return a tuple of (good, candidates) where candidates are "partially" sorted from trickiest considered
- **TODO** (*we might want to use it as a function where we would provide tdir*) –

`datalad.tests.utils_pytest.`**`get_mtimes_and_digests`**(*target_path*)

Return digests (md5) and mtimes for all the files under target_path

`datalad.tests.utils_pytest.`**`get_ssh_port`**(*host*)

Get port of *host* in ssh_config.

Our tests depend on the host being defined in ssh_config, including its port. This method can be used by tests that want to check handling of an explicitly specified

Note that if *host* does not match a host in ssh_config, the default value of 22 is returned.

Skips test if port cannot be found.

Parameters
host (*str*) –

Return type
port (int)

`datalad.tests.utils_pytest.`**`has_symlink_capability`**(*p1*, *p2*)

`datalad.tests.utils_pytest.`**`ignore_nose_capturing_stdout`**(*func*)

DEPRECATED and will be removed soon. Does nothing!

Originally was intended as a decorator workaround for nose's behaviour with redirecting sys.stdout, but now we monkey patch nose now so no test should no longer be skipped.

See issue reported here: https://code.google.com/p/python-nose/issues/detail?id=243&can=1&sort=-id&colspec=ID%20Type%20Status%20Priority%20Stars%20Milestone%20Owner%20Summary

`datalad.tests.utils_pytest.`**`in_`**(*first*, *second*, *msg=None*)

`datalad.tests.utils_pytest.`**`integration`**(*f*)

Mark test as an "integration" test which generally is not needed to be run

Generally tend to be slower. Should be used in combination with @slow and @turtle if that is the case.

`datalad.tests.utils_pytest.`**`known_failure`**(*func*)

Test decorator marking a test as known to fail

This combines *probe_known_failure* and *skip_known_failure* giving the skipping precedence over the probing.

`datalad.tests.utils_pytest.`**`known_failure_direct_mode`**(*func*)

DEPRECATED. Stop using. Does nothing

Test decorator marking a test as known to fail in a direct mode test run

If datalad.repo.direct is set to True behaves like *known_failure*. Otherwise the original (undecorated) function is returned.

datalad.tests.utils_pytest.**known_failure_githubci_osx**(*func*)

Test decorator for a known test failure on Github's macOS CI

datalad.tests.utils_pytest.**known_failure_githubci_win**(*func*)

Test decorator for a known test failure on Github's Windows CI

datalad.tests.utils_pytest.**known_failure_osx**(*func*)

Test decorator for a known test failure on macOS

datalad.tests.utils_pytest.**known_failure_windows**(*func*)

Test decorator marking a test as known to fail on windows

On Windows behaves like *known_failure*. Otherwise the original (undecorated) function is returned.

datalad.tests.utils_pytest.**maybe_adjust_repo**(*repo*)

Put repo into an adjusted branch if it is not already.

datalad.tests.utils_pytest.**neq_**(*first*, *second*, *msg=None*)

datalad.tests.utils_pytest.**nok_**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**nok_startswith**(*s*, *prefix*)

datalad.tests.utils_pytest.**ok_**(*expr*, *msg=None*)

datalad.tests.utils_pytest.**ok_annex_get**(*ar*, *files*, *network=True*)

Helper to run .get decorated checking for correct operation

get passes through stderr from the ar to the user, which pollutes screen while running tests

Note: Currently not true anymore, since usage of –json disables progressbars

datalad.tests.utils_pytest.**ok_archives_caches**(*repopath*, *n=1*, *persistent=None*)

Given a path to repository verify number of archives

Parameters

- **repopath** (*str*) – Path to the repository
- **n** (*int, optional*) – Number of archives directories to expect
- **persistent** (*bool or None, optional*) – If None – both persistent and not count.

datalad.tests.utils_pytest.**ok_broken_symlink**(*path*)

datalad.tests.utils_pytest.**ok_clean_git**(*path*, *annex=None*, *index_modified=[]*, *untracked=[]*)

Obsolete test helper. Use assert_repo_status() instead.

Still maps a few common cases to the new helper, to ease transition in extensions.

datalad.tests.utils_pytest.**ok_endswith**(*s*, *suffix*)

datalad.tests.utils_pytest.**ok_exists**(*path*)

datalad.tests.utils_pytest.**ok_file_has_content**(*path*, *content*, *strip=False*, *re_=False*, *decompress=False*, ***kwargs*)

Verify that file exists and has expected content

datalad.tests.utils_pytest.**ok_file_under_git**(*path*, *filename=None*, *annexed=False*)

Test if file is present and under git/annex control

If relative path provided, then test from current directory

datalad.tests.utils_pytest.**ok_generator**(*gen*)

datalad.tests.utils_pytest.**ok_git_config_not_empty**(*ar*)

Helper to verify that nothing rewritten the config file

datalad.tests.utils_pytest.**ok_good_symlink**(*path*)

datalad.tests.utils_pytest.**ok_startswith**(*s*, *prefix*)

datalad.tests.utils_pytest.**ok_symlink**(*path*)

Checks whether path is either a working or broken symlink

datalad.tests.utils_pytest.**patch_config**(*vars*)

Patch our config with custom settings. Returns mock.patch cm

Only the merged configuration from all sources (global, local, dataset) will be patched. Source-constrained patches (e.g. only committed dataset configuration) are not supported.

datalad.tests.utils_pytest.**probe_known_failure**(*func*)

Test decorator allowing the test to pass when it fails and vice versa

Setting config datalad.tests.knownfailures.probe to True tests, whether or not the test is still failing. If it's not, an AssertionError is raised in order to indicate that the reason for failure seems to be gone.

datalad.tests.utils_pytest.**put_file_under_git**(*path*, *filename=None*, *content=None*, *annexed=False*)

Place file under git/annex and return used Repo

datalad.tests.utils_pytest.**run_under_dir**(*func*, *newdir='.'*)

Decorator to run tests under another directory

It is somewhat ugly since we can't really chdir back to a directory which had a symlink in its path. So using this decorator has potential to move entire testing run under the dereferenced directory name – sideeffect.

The only way would be to instruct testing framework (i.e. nose in our case ATM) to run a test by creating a new process with a new cwd

datalad.tests.utils_pytest.**serve_path_via_http**(*tfunc*, **targs*, *use_ssl=False*, *auth=None*)

Decorator which serves content of a directory via http url

Parameters

- **path** (*str*) – Directory with content to serve.
- **use_ssl** (*bool*) – Flag whether to set up SSL encryption and return a HTTPS URL. This require a valid certificate setup (which is tested for proper function) or it will cause a SkipTest to be raised.
- **auth** (*tuple or None*) – If a (username, password) tuple is given, the server access will be protected via HTTP basic auth.

datalad.tests.utils_pytest.**set_annex_version**(*version*)

Override the git-annex version.

This temporarily masks the git-annex version present in external_versions and make AnnexRepo forget its cached version information.

datalad.tests.utils_pytest.**set_date**(*timestamp*)

Temporarily override environment variables for git/git-annex dates.

Parameters

timestamp (*int*) – Unix timestamp.

datalad.tests.utils_pytest.**skip_if**(*func*, *cond=True*, *msg=None*, *method='raise'*)

Skip test for specific condition

Parameters

- **cond** (*bool*) – condition on which to skip
- **msg** (*str*) – message to print if skipping
- **method** (*str*) – either 'raise' or 'pass'. Whether to skip by raising *SkipTest* or by just proceeding and simply not calling the decorated function. This is particularly meant to be used, when decorating single assertions in a test with method='pass' in order to not skip the entire test, but just that assertion.

datalad.tests.utils_pytest.**skip_if_adjusted_branch**(*func*)

Skip test if adjusted branch is used by default on TMPDIR file system.

datalad.tests.utils_pytest.**skip_if_no_module**(*module*)

datalad.tests.utils_pytest.**skip_if_no_network**(*func=None*)

Skip test completely in NONETWORK settings

If not used as a decorator, and just a function, could be used at the module level

datalad.tests.utils_pytest.**skip_if_on_windows**(*func=None*)

Skip test completely under Windows

datalad.tests.utils_pytest.**skip_if_root**(*func=None*)

Skip test if uid == 0.

Note that on Windows (or anywhere else *os.geteuid* is not available) the test is _not_ skipped.

datalad.tests.utils_pytest.**skip_if_scrapy_without_selector**()

A little helper to skip some tests which require recent scrapy

datalad.tests.utils_pytest.**skip_if_url_is_not_available**(*url*, *regex=None*)

datalad.tests.utils_pytest.**skip_known_failure**(*func*, *method='raise'*)

Test decorator allowing to skip a test that is known to fail

Setting config datalad.tests.knownfailures.skip to a bool enables/disables skipping.

datalad.tests.utils_pytest.**skip_nomultiplex_ssh**(*func*)

Skips SSH tests if default connection/manager does not support multiplexing

e.g. currently on windows or if set via datalad.ssh.multiplex-connections config variable

datalad.tests.utils_pytest.**skip_ssh**(*func*)

Skips SSH tests if on windows or if environment variable DATALAD_TESTS_SSH was not set

datalad.tests.utils_pytest.**skip_wo_symlink_capability**(*func*)

Skip test when environment does not support symlinks

Perform a behavioral test instead of top-down logic, as on windows this could be on or off on a case-by-case basis.

datalad.tests.utils_pytest.**slow**(*f*)

Mark test as a slow, although not necessarily integration or usecase test

Rule of thumb cut-off to mark as slow is 10 sec

datalad.tests.utils_pytest.**turtle**(*f*)

Mark test as very slow, meaning to not run it on Travis due to its time limit

Rule of thumb cut-off to mark as turtle is 2 minutes

datalad.tests.utils_pytest.**usecase**(*f*)

Mark test as a usecase user ran into and which (typically) caused bug report to be filed/troubleshooted

Should be used in combination with @slow and @turtle if slow.

datalad.tests.utils_pytest.**with_fake_cookies_db**(*func*, *cookies={}*)

mock original cookies db with a fake one for the duration of the test

datalad.tests.utils_pytest.**with_memory_keyring**(*t*)

Decorator to use non-persistent MemoryKeyring instance

datalad.tests.utils_pytest.**with_sameas_remote**(*func*, *autoenabled=False*)

Provide a repository with a git-annex sameas remote configured.

The repository will have two special remotes: r_dir (type=directory) and r_rsync (type=rsync). The rsync remote will be configured with –sameas=r_dir, and autoenabled if *autoenabled* is true.

datalad.tests.utils_pytest.**with_tempfile**(*t*, ***tkwargs*)

Decorator function to provide a temporary file name and remove it at the end

Parameters

- **set** (*To change the used directory without providing keyword argument 'dir'*) –
- **DATALAD_TESTS_TEMP_DIR.** –
- **mkdir** (*bool, optional (default: False)*) – If True, temporary directory created using tempfile.mkdtemp()
- **content** (*str or bytes, optional*) – Content to be stored in the file created
- **wrapped** (*function, optional*) – If set, function name used to prefix temporary file name
- ****tkwargs** – All other arguments are passed into the call to tempfile.mk{,d}temp(), and resultant temporary filename is passed as the first argument into the function t. If no 'prefix' argument is provided, it will be constructed using module and function names ('.' replaced with '_').

Examples

```
@with_tempfile
def test_write(tfile=None):
    open(tfile, 'w').write('silly test')
```

datalad.tests.utils_pytest.**with_testsui**(*t*, *responses=None*, *interactive=True*)

Switch main UI to be 'tests' UI and possibly provide answers to be used

datalad.tests.utils_pytest.**with_tree**(*t*, *tree=None*, *archives_leading_dir=True*, *delete=True*, ***tkwargs*)

datalad.tests.utils_pytest.**without_http_proxy**(*tfunc*)

Decorator to remove http*_proxy env variables for the duration of the test

datalad.tests.utils_testrepos

class datalad.tests.utils_testrepos.**BasicAnnexTestRepo**(*path=None*, *puke_if_exists=True*)

Bases: *TestRepo*

Creates a basic test git-annex repository

REPO_CLASS

alias of *AnnexRepo*

create_info_file()

populate()

class datalad.tests.utils_testrepos.**BasicGitTestRepo**(*path=None*, *puke_if_exists=True*)

Bases: *TestRepo*

Creates a basic test git repository.

REPO_CLASS

alias of *GitRepo*

create_info_file()

populate()

class datalad.tests.utils_testrepos.**InnerSubmodule**

Bases: object

create()

property path

property url

class datalad.tests.utils_testrepos.**NestedDataset**(*path=None*, *puke_if_exists=True*)

Bases: *BasicAnnexTestRepo*

populate()

class datalad.tests.utils_testrepos.**SubmoduleDataset**(*path=None*, *puke_if_exists=True*)

Bases: *BasicAnnexTestRepo*

populate()

class datalad.tests.utils_testrepos.**TestRepo**(*path=None*, *puke_if_exists=True*)

Bases: object

REPO_CLASS = None

create()

create_file(*name*, *content*, *add=True*, *annex=False*)

property path

abstract populate()

property url

datalad.tests.heavyoutput

Helper to provide heavy load on stdout and stderr

Command interface

interface.base	High-level interface generation

datalad.interface.base

High-level interface generation

class datalad.interface.base.**Interface**

Bases: ABC

Abstract base class for DataLad command implementations

Any DataLad command implementation must be derived from this class. The code snippet below shows a complete sketch of a Python class with such an implementation.

Importantly, no instances of command classes will created. Instead the main entry point is a static `__call__()` method, which must be implemented for any command. It is incorporated as a function in `datalad.api`, by default under the name of the file the implementation resides (e.g., `command` for a `command.py` file). Therefore the file should have a name that is a syntax-compliant function name. The default naming rule can be overwritten with an explicit alternative name (see *datalad.interface.base.get_api_name()*).

For commands implementing functionality that is operating on DataLad datasets, a command can be also be bound to the *Dataset* class as a method using the `@datasetmethod` decorator, under the specified name.

Any `__call__()` implementation should be decorated with `datalad.interface.utils.eval_results()`. This adds support for standard result processing, and a range of common command parameters that do not need to be manually added to the signature of `__call__()`. Any implementation decorated in this way should be implemented as a generator, and `yield` *result records*.

Any argument or keyword argument that appears in the signature of `__call__()` must have a matching item in `Interface._params_`. The dictionary maps argument names to `datalad.support.param.Parameter` specifications. The specification contain CLI argument declarations, value constraint and data type conversation specifications, documentation, and optional `argparse`-specific arguments for CLI parser construction.

The class decorator *datalad.interface.base.build_doc()* inspects an *Interface* implementation, and builds a standard docstring from various sources of structured information within the class (also see below). The documentation is automatically tuned differently, depending on the target API (Python vs CLI).

```
@build_doc
class ExampleCommand(Interface):
    """SHORT DESCRIPTION

    LONG DESCRIPTION
    ...
```

(continues on next page)

(continued from previous page)

```
    """

    # COMMAND PARAMETER DEFINITIONS
    _params_ = dict(
        example=Parameter(
            args=("--example",),
            doc="""Parameter description....""",
            constraints=...),
        ...
        )
    )

    # RESULT PARAMETER OVERRIDES
    return_type= 'list'
    ...

    # USAGE EXAMPLES
    _examples_ = [
        dict(text="Example description...",
             code_py="Example Python code...",
             code_cmd="Example shell code ..."),
        ...
    ]

    @staticmethod
    @datasetmethod(name='example_command')
    @eval_results
    def __call__(example=None, ...):
        ...

        yield dict(...)
```

The basic implementation setup described above can be customized for individual commands in various way that alter the behavior and presentation of a specific command. The following overview uses the code comment markers in the above snippet to illustrate where in the class implementation these adjustments can be made.

(SHORT/LONG) DESCRIPTION

`Interface.short_description` can be defined to provide an explicit short description to be used in documentation and help output, replacing the auto-generated extract from the first line of the full description.

COMMAND PARAMETER DEFINITIONS

When a parameter specification declares `Parameter(args=tuple(), ...)`, i.e. no arguments specified, it will be ignored by the CLI. Likewise, any `Parameter` specification for which *`is_api_arg()`* returns `False` will also be ignored by the CLI. Additionally, any such parameter will not be added to the parameter description list in the Python docstring.

RESULT PARAMETER OVERRIDES

The `datalad.interface.utils.eval_results()` decorator automatically add a range of additional arguments to a command, which are defined in `datalad.interface.common_opts.eval_params`. For any such parameter an Interface implementation can define an interface-specific default value, by declaring a class member with the respective parameter name and the desired default as its assigned value. This feature can be used to tune the default command behavior, for example, with respect to the default result rendering style, or its error behavior.

In addition to the common parameters of the Python API, an additional `Interface.result_renderer_cmdline` can be defined, in order to instruct the CLI to prefer the specified alternative result renderer over an `Interface.result_renderer` specification.

USAGE EXAMPLES

Any number of usage examples can be described in an `_examples_` list class attribute. Such an example contains a description, and code examples for Python and CLI.

classmethod get_refds_path(*dataset*)

Return a resolved reference dataset path from a *dataset* argument

Deprecated since version 0.16: Use `require_dataset()` instead.

on_failure = 'continue'

result_filter = None

result_renderer = 'tailored'

result_xfm = None

return_type = 'list'

datalad.interface.base.**alter_interface_docs_for_api**(*docs*)

Apply modifications to interface docstrings for Python API use.

datalad.interface.base.**build_doc**(*cls*, ***kwargs*)

Decorator to build docstrings for datalad commands

It's intended to decorate the class, the __call__-method of which is the actual command. It expects that __call__-method to be decorated by eval_results.

Note that values for any *eval_params* keys in *cls._params_* are ignored. This means one class may extend another's *_params_* without worrying about filtering out *eval_params*.

Parameters
cls (Interface) – DataLad command implementation

datalad.interface.base.**build_example**(*example*, *api='python'*)

Build a code example.

Take a dict from a classes _example_ specification (list of dicts) and build a string with an api or cmd example (for use in cmd help or docstring).

Parameters
api (*{'python', 'cmdline'}*) – If 'python', build Python example for docstring. If 'cmdline', build cmd example.

Returns
ex – Concatenated examples for the given class.

Return type
str

datalad.interface.base.**dedent_docstring**(*text*)

Remove uniform indentation from a multiline docstring

datalad.interface.base.**eval_results**(*wrapped*)

Decorator for return value evaluation of datalad commands.

Note, this decorator is only compatible with commands that return status dict sequences!

Two basic modes of operation are supported: 1) "generator mode" that *yields* individual results, and 2) "list mode" that returns a sequence of results. The behavior can be selected via the kwarg *return_type*. Default is "list mode".

This decorator implements common functionality for result rendering/output, error detection/handling, and logging.

Result rendering/output configured via the *result_renderer* keyword argument of each decorated command. Supported modes are: 'generic' (a generic renderer producing one line per result 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), 'tailored' custom output formatting provided by each command class (if any), or 'disabled' for no result rendering.

Error detection works by inspecting the *status* item of all result dictionaries. Any occurrence of a status other than 'ok' or 'notneeded' will cause an IncompleteResultsError exception to be raised that carries the failed actions' status dictionaries in its *failed* attribute.

Status messages will be logged automatically, by default the following association of result status and log channel will be used: 'ok' (debug), 'notneeded' (debug), 'impossible' (warning), 'error' (error). Logger instances included in the results are used to capture the origin of a status report.

Parameters

func (*function*) – __call__ method of a subclass of Interface, i.e. a datalad command definition

datalad.interface.base.**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

datalad.interface.base.**get_api_name**(*intfspec*)

Given an interface specification return an API name for it

datalad.interface.base.**get_cmd_doc**(*interface*)

Return the documentation for the command defined by *interface*.

Parameters

interface (*subclass of* Interface) –

datalad.interface.base.**get_cmd_summaries**(*descriptions*, *groups*, *width=79*)

Return summaries for the commands in *groups*.

Parameters

- **descriptions** (*dict*) – A map of group names to summaries.
- **groups** (*list of tuples*) – A list of groups and commands in the form described by *get_interface_groups*.
- **width** (*int, optional*) – The maximum width of each line in the summary text.

Returns

- *A list with a formatted entry for each command. The first command of each*
- *group is preceded by an entry describing the group.*

datalad.interface.base.**get_interface_groups**(*include_plugins=False*)

Return a list of command groups.

Return type
A list of tuples with the form (GROUP_NAME, GROUP_DESCRIPTION, COMMANDS).

datalad.interface.base.**is_api_arg**(*arg*)

Return True if argument is our API argument or self or used for internal purposes

datalad.interface.base.**load_interface**(*spec*)

Load and return the class for *spec*.

Parameters
spec (*tuple*) – For a standard interface, the first item is the datalad source module and the second object name for the interface.

Return type
The interface class or, if importing the module fails, None.

datalad.interface.base.**update_docstring_with_examples**(*cls_doc*, *ex*)

Update a commands docstring with examples.

Take _examples_ of a command, build the Python examples, and append them to the docstring.

Parameters

- **cls_doc** (*str*) – docstring
- **ex** (*list*) – list of dicts with examples

datalad.interface.base.**update_docstring_with_parameters**(*func*, *params*, *prefix=None*, *suffix=None*, *add_args=None*)

Generate a useful docstring from a parameter spec

Amends any existing docstring of a callable with a textual description of its parameters. The Parameter spec needs to match the number and names of the callables arguments.

Command line interface infrastructure

cli.exec	Call a command interface
cli.main	This is the main() CLI entryproint
cli.parser	Components to build the parser instance for the CLI
cli.renderer	Render results in a terminal

datalad.cli.exec

Call a command interface

Provide a callable to register in a cmdline parser, for executing a parameterized command call.

datalad.cli.exec.**call_from_parser**(*cls*, *args*)

Executable to be registered with the parser for a particular command

Parameters

- **cls** (*Interface*) – Class implementing a particular interface.
- **args** (*Namespace*) – Populated argparse namespace instance.

Returns
Returns the iterable return by an command's implementation of `__call__()`. It is unwound, in case of a generator being returned to actually trigger the underlying processing.

Return type
iterable

datalad.cli.main

This is the main() CLI entryproint

`datalad.cli.main.main(`*args=['/home/docs/checkouts/readthedocs.org/user_builds/datalad/envs/latest/lib/python3.9/site-packages/sphinx/__main__.py', '-T', '-b', 'html', '-d', '_build/doctrees', '-D', 'language=en', '.', '/home/docs/checkouts/readthedocs.org/user_builds/datalad/checkouts/latest/_readthedocs//html']*`)`

Main CLI entrypoint

datalad.cli.parser

Components to build the parser instance for the CLI

This module must import (and run) really fast for a responsive CLI. It is unconditionally imported by the main() entrypoint.

`class datalad.cli.parser.ArgumentParserDisableAbbrev(`*prog=None, usage=None, description=None, epilog=None, parents=[], formatter_class=<class 'argparse.HelpFormatter'>, prefix_chars='-', fromfile_prefix_chars=None, argument_default=None, conflict_handler='error', add_help=True, allow_abbrev=True, exit_on_error=True*`)`

Bases: `ArgumentParser`

`datalad.cli.parser.add_subparser(`*_intfspec, subparsers, cmd_name, formatter_class, completing=False*`)`

Given an interface spec, add a subparser to subparsers under cmd_name

`datalad.cli.parser.fail_with_short_help(`*parser=None, msg=None, known=None, provided=None, hint=None, exit_code=1, what='command', out=None*`)`

Generic helper to fail with short help possibly hinting on what was intended if *known* were provided

`datalad.cli.parser.parser_add_common_opt(`*parser, opt, names=None, **kwargs*`)`

`datalad.cli.parser.parser_add_common_options(`*parser, version=None*`)`

Add all options defined in common_args, but excludes 'help'

`datalad.cli.parser.parser_add_version_opt(`*parser, mod_name, include_name=False, delay=False*`)`

Setup –version option

Parameters

- **parser** –
- **mod_name** (*str, optional*) –
- **include_name** (*bool, optional*) –

- **delay** (*bool, optional*) – If set to True, no action is taken immediately, and rather we assign the function which would print the version. Necessary for early pre-parsing of the cmdline

datalad.cli.parser.**setup_parser**(*cmdlineargs*, *formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>*, *return_subparsers=False*, *completing=False*, *help_ignore_extensions=False*)

The holy grail of establishing CLI for DataLad's Interfaces

Parameters

- **cmdlineargs** (*sys.argv*) – Used to make some shortcuts when construction of a full parser can be avoided.
- **formatter_class** – Passed to argparse
- **return_subparsers** (*bool, optional*) – is used ATM only by BuildManPage in _datalad_build_support
- **completing** (*bool, optional*) – Flag to indicate whether the process was invoked by argcomplete
- **help_ignore_extensions** (*bool, optional*) – Prevent loading of extension entrypoints when –help is requested. This is enabled when building docs to avoid pollution of generated manpages with extensions commands (that should appear in their own docs, but not in the core datalad package docs)

datalad.cli.parser.**setup_parser_for_interface**(*parser*, *cls*, *completing=False*)

datalad.cli.parser.**setup_parserarg_for_interface**(*parser*, *param_name*, *param*, *defaults_idx*, *prefix_chars*, *defaults*, *completing=False*)

datalad.cli.parser.**single_subparser_possible**(*cmdlineargs*, *parser*, *completing*)

Performs early analysis of the cmdline

Looks at the first unparsed argument and if a known command, would return only that one.

When a plain command invocation with *–version* is detected, it will be acted on directly (until sys.exit(0) to avoid wasting time on unnecessary further processing.

Returns

Returns a status label and a parameter for this status. 'error': parsing failed, 'allknown': the parser successfully identified all arguments, 'help': a help request option was found, 'unknownopt': an unknown or incomplete option was found, 'subcommand': a potential subcommand name was found. For the latter two modes the second return value is the option or command name. For all other modes the second return value is None.

Return type

{'error', 'allknown', 'help', 'unknownopt', 'subcommand'}, None or str

datalad.cli.parser.**try_suggest_extension_with_command**(*parser*, *cmd*, *completing*, *known_cmds*)

If completing=False, this function will trigger sys.exit()

datalad.cli.renderer

Render results in a terminal

class datalad.cli.renderer.**DefaultOutputFormatter**(*missing=nagen()*)

Bases: Formatter

A custom formatter for default output rendering using .format

get_value(*key*, *args*, *kwds*)

class datalad.cli.renderer.**DefaultOutputRenderer**(*format*)

Bases: object

A default renderer for .format'ed output line

datalad.cli.renderer.**nadict**(**items*)

A generator of default dictionary with the default nagen

class datalad.cli.renderer.**nagen**(*missing='N/A'*)

Bases: object

A helper to provide a desired missing value if no value is known

Usecases

- could be used as a generator for *defaultdict*
- since it returns itself upon getitem, should work even for complex nested dictionaries/lists .format templates

1.5.3 Configuration

DataLad uses the same configuration mechanism and syntax as Git itself. Consequently, datalad can be configured using the **git config** command. Both a *global* user configuration (typically at ~/.gitconfig), and a *local* repository-specific configuration (.git/config) are inspected.

In addition, datalad supports a persistent dataset-specific configuration. This configuration is stored at .datalad/config in any dataset. As it is part of a dataset, settings stored there will also be in effect for any consumer of such a dataset. Both *global* and *local* settings on a particular machine always override configuration shipped with a dataset.

All datalad-specific configuration variables are prefixed with datalad..

It is possible to override or amend the configuration using environment variables. Any variable with a name that starts with DATALAD_ will be available as the corresponding datalad. configuration variable, replacing any __ (two underscores) with a hyphen, then any _ (single underscore) with a dot, and finally converting all letters to lower case. Values from environment variables take precedence over configuration file settings.

In addition, the DATALAD_CONFIG_OVERRIDES_JSON environment variable can be set to a JSON record with configuration values. This is particularly useful for options that aren't accessible through the naming scheme described above (e.g., an option name that includes an underscore).

The following sections provide a (non-exhaustive) list of settings honored by datalad. They are categorized according to the scope they are typically associated with.

Global user configuration

datalad.clone.url-substitute.github
GitHub URL substitution rule: Mangling for GitHub-related URL. A substitution specification is a string with a match and substitution expression, each following Python's regular expression syntax. Both expressions are concatenated to a single string with an arbitrary delimiter character. The delimiter is defined by prefixing the string with the delimiter. Prefix and delimiter are stripped from the expressions (Example: ",^http://(.*)$,https://1"). This setting can be defined multiple times. Substitutions will be applied incrementally, in order of their definition. The first substitution in such a series must match, otherwise no further substitutions in a series will be considered. However, following the first match all further substitutions in a series are processed, regardless whether intermediate expressions match or not. Default: (',https?://github.com/([^/]+)/(.*)$,\1###\2', ',[/\\]+(?!$),-', ',\s+|(%2520)+|(%20)+,_', ',([^#]+)###(.*),https://github.com/\1/\2')

datalad.clone.url-substitute.osf
Open Science Framework URL substitution rule: Mangling for OSF-related URLs. A substitution specification is a string with a match and substitution expression, each following Python's regular expression syntax. Both expressions are concatenated to a single string with an arbitrary delimiter character. The delimiter is defined by prefixing the string with the delimiter. Prefix and delimiter are stripped from the expressions (Example: ",^http://(.*)$,https://1"). This setting can be defined multiple times. Substitutions will be applied incrementally, in order of their definition. The first substitution in such a series must match, otherwise no further substitutions in a series will be considered. However, following the first match all further substitutions in a series are processed, regardless whether intermediate expressions match or not. Default: (',^https://osf.io/([^/]+)[/]*$,osf://\1',)

datalad.extensions.load
DataLad extension packages to load: Indicate which extension packages should be loaded unconditionally on CLI startup or on importing 'datalad.[core]api'. This enables the respective extensions to customize DataLad with functionality and configurability outside the scope of extension commands. For merely running extension commands it is not necessary to load them specifically Default: None

datalad.externals.nda.dbserver
NDA database server: Hostname of the database server Default: https://nda.nih.gov/DataManager/dataManager

datalad.locations.cache
Cache directory: Where should datalad cache files? Default: ~/.cache/datalad

datalad.locations.default-dataset
Default dataset path: Where should datalad should look for (or install) a default dataset? Default: ~/datalad

datalad.locations.extra-procedures
Extra procedure directory: Where should datalad search for some additional procedures?

datalad.locations.locks
Lockfile directory: Where should datalad store lock files? Default: ~/.cache/datalad/locks

datalad.locations.sockets
Socket directory: Where should datalad store socket files? Default: ~/.cache/datalad/sockets

datalad.locations.system-procedures
System procedure directory: Where should datalad search for system procedures? Default: /etc/xdg/datalad/procedures

datalad.locations.user-procedures
User procedure directory: Where should datalad search for user procedures? Default: ~/.config/datalad/procedures

datalad.ssh.executable
Name of ssh executable for 'datalad sshrun': Specifies the name of the ssh-client executable thatdatalad will use. This might be an absolute path. On Windows systems it is currently by default set to point to the ssh executable

of OpenSSH for Windows, if OpenSSH for Windows is installed. On other systems it defaults to 'ssh'. Default: ssh

[value must be a string]

datalad.ssh.identityfile
If set, pass this file as ssh's -i option.: Default: None

datalad.ssh.multiplex-connections
Whether to use a single shared connection for multiple SSH processes aiming at the same target.: Default: True

[value must be convertible to type bool]

datalad.ssh.try-use-annex-bundled-git
Whether to attempt adjusting the PATH in a remote shell to include Git binaries located in a detected git-annex bundle: If enabled, this will be a 'best-effort' attempt that only supports remote hosts with a Bourne shell and the *which* command available. The remote PATH must already contain a git-annex installation. If git-annex is not found, or the detected git-annex does not have a bundled Git installation, detection failure will not result in an error, but only slow remote execution by one-time sensing overhead per each opened connection. Default: False

[value must be convertible to type bool]

datalad.tests.cache
Cache directory for tests: Where should datalad cache test files? Default: ~/.cache/datalad/tests

datalad.tests.credentials
Credentials to use during tests: Which credentials should be available while running tests? If "plaintext" (default), a new plaintext keyring would be created in tests temporary HOME. If "system", no custom configuration would be passed to keyring and known to system credentials could be used. Default: plaintext

[value must be one of [CMD: ('plaintext', 'system') CMD][PY: ('plaintext', 'system') PY]]

Local repository configuration

datalad.crawl.cache
Crawler download caching: Should the crawler cache downloaded files?

[value must be convertible to type bool]

datalad.fake-dates
Fake (anonymize) dates: Should the dates in the logs be faked? Default: False

[value must be convertible to type bool]

Sticky dataset configuration

datalad.locations.dataset-procedures
Dataset procedure directory: Where should datalad search for dataset procedures (relative to a dataset root)? Default: .datalad/procedures

Miscellaneous configuration

datalad.annex.retry
Value for annex.retry to use for git-annex calls: On transfer failure, annex.retry (sans "datalad.") controls the number of times that git-annex retries. DataLad will call git-annex with annex.retry set to the value here unless the annex.retry is explicitly configured Default: 3

[value must be convertible to type 'int']

datalad.credentials.force-ask
Force (re-)entry of credentials: Should DataLad prompt for credential (re-)entry? This can be used to update previously stored credentials. Default: False

[value must be convertible to type bool]

datalad.credentials.githelper.noninteractive
Non-interactive mode for git-credential helper: Should git-credential-datalad operate in non-interactive mode? This would mean to not ask for user confirmation when storing new credentials/provider configs. Default: False

[bool]

datalad.exc.str.tblimit
This flag is used by datalad to cap the number of traceback steps included in exception logging and result reporting to DATALAD_EXC_STR_TBLIMIT of pre-processed entries from traceback.:

datalad.fake-dates-start
Initial fake date: When faking dates and there are no commits in any local branches, generate the date by adding one second to this value (Unix epoch time). The value must be positive. Default: 1112911993

[value must be convertible to type 'int']

datalad.github.token-note
GitHub token note: Description for a Personal access token to generate. Default: DataLad

datalad.install.inherit-local-origin
Inherit local origin of dataset source: If enabled, a local 'origin' remote of a local dataset clone source is configured as an 'origin-2' remote to make its annex automatically available. The process is repeated recursively for any further qualifying 'origin' dataset thereof.Note that if clone.defaultRemoteName is configured to use a name other than 'origin', that name will be used instead. Default: True

[value must be convertible to type bool]

datalad.log.level
Used for control the verbosity of logs printed to stdout while running datalad commands/debugging:

datalad.log.name
Include name of the log target in the log line:

datalad.log.names
Which names (,-separated) to print log lines for:

datalad.log.namesre
Regular expression for which names to print log lines for:

datalad.log.outputs
Whether to log stdout and stderr for executed commands: When enabled, setting the log level to 5 should catch all execution output, though some output may be logged at higher levels Default: False

[value must be convertible to type bool]

datalad.log.result-level
Log level for command result messages: If 'match-status', it will log 'impossible' results as a warning, 'error'

results as errors, and everything else as 'debug'. Otherwise the indicated log-level will be used for all such messages Default: debug

[value must be one of [CMD: ('debug', 'info', 'warning', 'error', 'match-status') CMD][PY: ('debug', 'info', 'warning', 'error', 'match-status') PY]]

datalad.log.timestamp
Used to add timestamp to datalad logs: Default: False

[value must be convertible to type bool]

datalad.log.traceback
Includes a compact traceback in a log message, with generic components removed. This setting is only in effect when given as an environment variable DATALAD_LOG_TRACEBACK. An integer value specifies the maximum traceback depth to be considered. If set to "collide", a common traceback prefix between a current traceback and a previously logged traceback is replaced with "…" (maximum depth 100).:

datalad.repo.backend
git-annex backend: Backend to use when creating git-annex repositories Default: MD5E

datalad.repo.direct
Direct Mode for git-annex repositories: Set this flag to create annex repositories in direct mode by default Default: False

[value must be convertible to type bool]

datalad.repo.version
git-annex repository version: Specifies the repository version for git-annex to be used by default Default: 8

[value must be convertible to type 'int']

datalad.runtime.max-annex-jobs
Maximum number of git-annex jobs to request when "jobs" option set to "auto" (default): Set this value to enable parallel annex jobs that may speed up certain operations (e.g. get file content). The effective number of jobs will not exceed the number of available CPU cores (or 3 if there is less than 3 cores). Default: 1

[value must be convertible to type 'int']

datalad.runtime.max-batched
Maximum number of batched commands to run in parallel: Automatic cleanup of batched commands will try to keep at most this many commands running. Default: 20

[value must be convertible to type 'int']

datalad.runtime.max-inactive-age
Maximum time (in seconds) a batched command can be inactive before it is eligible for cleanup: Automatic cleanup of batched commands will consider an inactive command eligible for cleanup if more than this many seconds have transpired since the command's last activity. Default: 60

[value must be convertible to type 'int']

datalad.runtime.max-jobs
Maximum number of jobs DataLad can run in "parallel": Set this value to enable parallel multi-threaded DataLad jobs that may speed up certain operations, in particular operation across multiple datasets (e.g., install multiple subdatasets, etc). Default: 1

[value must be convertible to type 'int']

datalad.runtime.pathspec-from-file
Provide list of files to git commands via –pathspec-from-file: Instructs when DataLad will provide list of paths to 'git' commands which support –pathspec-from-file option via some temporary file. If set to 'multi-chunk' it will be done only if multiple invocations of the command on chunks of files list is needed. If set to 'always', DataLad will always use –pathspec-from-file. Default: multi-chunk

[value must be one of [CMD: ('multi-chunk', 'always') CMD][PY: ('multi-chunk', 'always') PY]]

datalad.runtime.raiseonerror
Error behavior: Set this flag to cause DataLad to raise an exception on errors that would have otherwise just get logged Default: False

[value must be convertible to type bool]

datalad.runtime.report-status
Command line result reporting behavior: If set (to other than 'all'), constrains command result report to records matching the given status. 'success' is a synonym for 'ok' OR 'notneeded', 'failure' stands for 'impossible' OR 'error' Default: None

[value must be one of [CMD: ('all', 'success', 'failure', 'ok', 'notneeded', 'impossible', 'error') CMD][PY: ('all', 'success', 'failure', 'ok', 'notneeded', 'impossible', 'error') PY]]

datalad.runtime.stalled-external
Behavior for handing external processes: What to do with external processes if they do not finish in some minimal reasonable time. If "abandon", datalad would proceed without waiting for external process to exit. ATM applies only to batched git-annex processes. Should be changed with caution. Default: wait

[value must be one of [CMD: ('wait', 'abandon') CMD][PY: ('wait', 'abandon') PY]]

datalad.save.no-message
Commit message handling: When no commit message was provided: attempt to obtain one interactively (interactive); or use a generic commit message (generic). NOTE: The interactive option is experimental. The behavior may change in backwards-incompatible ways. Default: generic

[value must be one of [CMD: ('interactive', 'generic') CMD][PY: ('interactive', 'generic') PY]]

datalad.save.windows-compat-warning
Action when Windows-incompatible file names are saved: Certain characters or names can make file names incompatible with Windows. If such files are saved 'warning' will alert users with a log message, 'error' will yield an 'impossible' result, and 'none' will ignore the incompatibility. Default: warning

[value must be one of [CMD: ('warning', 'error', 'none') CMD][PY: ('warning', 'error', 'none') PY]]

datalad.source.epoch
Datetime epoch to use for dates in built materials: Datetime to use for reproducible builds. Originally introduced for Debian packages to interface SOURCE_DATE_EPOCH described at https://reproducible-builds.org/docs/source-date-epoch/ .By default - current time Default: 1709821133.893068

[value must be convertible to type 'float']

datalad.tests.dataladremote
Binary flag to specify whether each annex repository should get datalad special remote in every test repository:

[value must be convertible to type bool]

datalad.tests.knownfailures.probe
Probes tests that are known to fail on whether or not they are actually still failing: Default: False

[value must be convertible to type bool]

datalad.tests.knownfailures.skip
Skips tests that are known to currently fail: Default: True

[value must be convertible to type bool]

datalad.tests.nonetwork
Skips network tests completely if this flag is set, Examples include test for S3, git_repositories, OpenfMRI, etc:

[value must be convertible to type bool]

datalad.tests.nonlo
Specifies network interfaces to bring down/up for testing. Currently used by Travis CI.:

datalad.tests.noteardown
Does not execute teardown_package which cleans up temp files and directories created by tests if this flag is set:

[value must be convertible to type bool]

datalad.tests.runcmdline
Binary flag to specify if shell testing using shunit2 to be carried out:

[value must be convertible to type bool]

datalad.tests.setup.testrepos
Pre-creates repositories for @with_testrepos within setup_package: Default: False

[value must be convertible to type bool]

datalad.tests.ssh
Skips SSH tests if this flag is **not** set:

[value must be convertible to type bool]

datalad.tests.temp.dir
Create a temporary directory at location specified by this flag. It is used by tests to create a temporary git directory while testing git annex archives etc: Default: None

[value must be a string]

datalad.tests.temp.fs
Specify the temporary file system to use as loop device for testing DATALAD_TESTS_TEMP_DIR creation:

datalad.tests.temp.fssize
Specify the size of temporary file system to use as loop device for testing DATALAD_TESTS_TEMP_DIR creation:

datalad.tests.temp.keep
Function rmtemp will not remove temporary file/directory created for testing if this flag is set:

[value must be convertible to type bool]

datalad.tests.ui.backend
Tests UI backend: Which UI backend to use Default: tests-noninteractive

datalad.tests.usecassette
Specifies the location of the file to record network transactions by the VCR module. Currently used by when testing custom special remotes:

datalad.ui.color
Colored terminal output: Enable or disable ANSI color codes in outputs; "on" overrides NO_COLOR environment variable Default: auto

[value must be one of [CMD: ('on', 'off', 'auto') CMD][PY: ('on', 'off', 'auto') PY]]

datalad.ui.progressbar
UI progress bars: Default backend for progress reporting Default: None

[value must be one of [CMD: ('tqdm', 'tqdm-ipython', 'log', 'none') CMD][PY: ('tqdm', 'tqdm-ipython', 'log', 'none') PY]]

datalad.ui.suppress-similar-results
Suppress rendering of similar repetitive results: If enabled, after a certain number of subsequent results that are identical regarding key properties, such as 'status', 'action', and 'type', additional similar results are not rendered

by the common result renderer anymore. Instead, a count of suppressed results is displayed. If disabled, or when not running in an interactive terminal, all results are rendered. Default: True

[value must be convertible to type bool]

datalad.ui.suppress-similar-results-threshold
Threshold for suppressing similar repetitive results: Minimum number of similar results to occur before suppression is considered. See 'datalad.ui.suppress-similar-results' for more information. Default: 10

[value must be convertible to type 'int']

1.6 Extension packages

DataLad can be customized and additional functionality can be integrated via extensions. Each extension provides its own documentation:

- Crawling web resources and automated data distributions
- Neuroimaging data and workflows
- Containerized computational environments
- Advanced metadata tooling with JSON-LD reporting and additional metadata extractors
- Staged additions, performance and user experience improvements for DataLad
- Resources for working with the UKBiobank as a DataLad dataset
- Deposit and retrieve DataLad datasets via the Open Science Framework
- Functionality that has been phased out of the core package
- Special interest functionality or drafts of future additions to DataLad proper

1.7 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

datalad.cli.exec, 435
datalad.cli.main, 436
datalad.cli.parser, 436
datalad.cli.renderer, 438
datalad.cmd, 337
datalad.config, 416
datalad.consts, 340
datalad.customremotes.archives, 409
datalad.customremotes.base, 407
datalad.interface.base, 431
datalad.log, 340
datalad.runner.nonasyncrunner, 411
datalad.runner.protocol, 414
datalad.support.annexrepo, 387
datalad.support.archives, 405
datalad.support.extensions, 406
datalad.support.gitrepo, 367
datalad.tests.heavyoutput, 431
datalad.tests.utils_pytest, 422
datalad.tests.utils_testrepos, 430
datalad.utils, 343
datalad.version, 367

INDEX

Symbols

__init__() (*datalad.api.Dataset method*), 249

A

add() (*datalad.config.ConfigManager method*), 416
add() (*datalad.support.annexrepo.AnnexRepo method*), 387
add() (*datalad.support.gitrepo.GitRepo method*), 369
add_() (*datalad.support.annexrepo.AnnexRepo method*), 388
add_() (*datalad.support.gitrepo.GitRepo method*), 369
add_archive_content() (*in module datalad.api*), 319
add_fake_dates() (*datalad.support.gitrepo.GitRepo method*), 370
add_readme() (*in module datalad.api*), 321
add_remote() (*datalad.support.gitrepo.GitRepo method*), 370
add_subparser() (*in module datalad.cli.parser*), 436
add_to_output() (*datalad.support.annexrepo.AnnexJsonProtocol method*), 387
add_to_output() (*datalad.support.annexrepo.GeneratorAnnexJsonProtocol method*), 404
add_url_to_file() (*datalad.support.annexrepo.AnnexRepo method*), 388
add_urls() (*datalad.support.annexrepo.AnnexRepo method*), 388
addurls() (*in module datalad.api*), 322
adjust() (*datalad.support.annexrepo.AnnexRepo method*), 389
alter_interface_docs_for_api() (*in module datalad.interface.base*), 433
annex, **141**
AnnexCustomRemote (*class in datalad.customremotes.base*), 407
AnnexInitOutput (*class in datalad.support.annexrepo*), 387
AnnexJsonProtocol (*class in datalad.support.annexrepo*), 387
AnnexRepo (*class in datalad.support.annexrepo*), 387
annexstatus() (*datalad.support.annexrepo.AnnexRepo method*), 389
any_re_search() (*in module datalad.utils*), 345
anything2bool() (*in module datalad.config*), 420
ArchiveAnnexCustomRemote (*class in datalad.customremotes.archives*), 409
ArchivesCache (*class in datalad.support.archives*), 405
args (*datalad.utils.ArgSpecFake attribute*), 343
ArgSpecFake (*class in datalad.utils*), 343
ArgumentParserDisableAbbrev (*class in datalad.cli.parser*), 436
assert_cwd_unchanged() (*in module datalad.tests.utils_pytest*), 422
assert_dict_equal() (*in module datalad.tests.utils_pytest*), 422
assert_equal() (*in module datalad.tests.utils_pytest*), 422
assert_false() (*in module datalad.tests.utils_pytest*), 423
assert_greater() (*in module datalad.tests.utils_pytest*), 423
assert_greater_equal() (*in module datalad.tests.utils_pytest*), 423
assert_in() (*in module datalad.tests.utils_pytest*), 423
assert_in_results() (*in module datalad.tests.utils_pytest*), 423
assert_is() (*in module datalad.tests.utils_pytest*), 423
assert_is_generator() (*in module datalad.tests.utils_pytest*), 423
assert_is_instance() (*in module datalad.tests.utils_pytest*), 423
assert_is_none() (*in module datalad.tests.utils_pytest*), 423
assert_is_not() (*in module datalad.tests.utils_pytest*), 423
assert_is_not_none() (*in module datalad.tests.utils_pytest*), 423
assert_logged() (*datalad.utils.SwallowLogsAdapter method*), 344
assert_message() (*in module datalad.tests.utils_pytest*), 423
assert_no_errors_logged() (*in module data-*

lad.tests.utils_pytest), 423
assert_no_open_files() (*in module datalad.utils*), 345
assert_not_equal() (*in module datalad.tests.utils_pytest*), 423
assert_not_in() (*in module datalad.tests.utils_pytest*), 423
assert_not_in_results() (*in module datalad.tests.utils_pytest*), 423
assert_not_is_instance() (*in module datalad.tests.utils_pytest*), 423
assert_re_in() (*in module datalad.tests.utils_pytest*), 423
assert_repo_status() (*in module datalad.tests.utils_pytest*), 423
assert_result_count() (*in module datalad.tests.utils_pytest*), 424
assert_result_values_cond() (*in module datalad.tests.utils_pytest*), 424
assert_result_values_equal() (*in module datalad.tests.utils_pytest*), 424
assert_set_equal() (*in module datalad.tests.utils_pytest*), 424
assert_status() (*in module datalad.tests.utils_pytest*), 424
assert_str_equal() (*in module datalad.tests.utils_pytest*), 424
assert_true() (*in module datalad.tests.utils_pytest*), 424
assure_bool() (*in module datalad.utils*), 345
assure_bytes() (*in module datalad.utils*), 345
assure_dict_from_str() (*in module datalad.utils*), 345
assure_dir() (*in module datalad.utils*), 345
assure_extracted() (*datalad.support.archives.ExtractedArchive method*), 405
assure_iter() (*in module datalad.utils*), 346
assure_list() (*in module datalad.utils*), 346
assure_list_from_str() (*in module datalad.utils*), 346
assure_tuple_or_list() (*in module datalad.utils*), 346
assure_unicode() (*in module datalad.utils*), 346
attr() (*in module datalad.tests.utils_pytest*), 424
auto_repr() (*in module datalad.utils*), 347
AVAILABILITY (*datalad.customremotes.base.AnnexCustomRemote attribute*), 407

B

bare (*datalad.support.gitrepo.GitRepo property*), 370
BasicAnnexTestRepo (*class in datalad.tests.utils_testrepos*), 430
BasicGitTestRepo (*class in datalad.tests.utils_testrepos*), 430
BatchedAnnex (*class in datalad.support.annexrepo*), 403
BatchedAnnexes (*class in datalad.support.annexrepo*), 404
BatchedCommand (*class in datalad.cmd*), 337
BatchedCommandError, 339
BatchedCommandProtocol (*class in datalad.cmd*), 339
BEGIN (*datalad.support.gitrepo.GitProgress attribute*), 368
build_doc() (*in module datalad.interface.base*), 433
build_example() (*in module datalad.interface.base*), 433
bytes2human() (*in module datalad.utils*), 347

C

cache (*datalad.customremotes.archives.ArchiveAnnexCustomRemote property*), 409
call_annex() (*datalad.support.annexrepo.AnnexRepo method*), 389
call_annex_items_() (*datalad.support.annexrepo.AnnexRepo method*), 389
call_annex_oneline() (*datalad.support.annexrepo.AnnexRepo method*), 389
call_annex_records() (*datalad.support.annexrepo.AnnexRepo method*), 390
call_annex_success() (*datalad.support.annexrepo.AnnexRepo method*), 390
call_from_parser() (*in module datalad.cli.exec*), 435
check_dates() (*in module datalad.api*), 326
check_direct_mode_support() (*datalad.support.annexrepo.AnnexRepo class method*), 390
check_for_stall() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
check_not_generatorfunction() (*in module datalad.tests.utils_pytest*), 424
check_repository_versions() (*datalad.support.annexrepo.AnnexRepo class method*), 390
check_symlink_capability() (*in module datalad.utils*), 347
CHECKING_OUT (*datalad.support.gitrepo.GitProgress attribute*), 368
checkout() (*datalad.support.gitrepo.GitRepo method*), 370
checkpresent() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote*

method), 409
checkurl() (datalad.customremotes.archives.ArchiveAnnexCustomRemote method), 409
cherry_pick() (datalad.support.gitrepo.GitRepo method), 370
chpwd (class in datalad.utils), 348
claimurl() (datalad.customremotes.archives.ArchiveAnnexCustomRemote method), 410
clean() (datalad.support.archives.ArchivesCache method), 405
clean() (datalad.support.archives.ExtractedArchive method), 405
clean() (in module datalad.api), 300
clean_inactive() (datalad.cmd.BatchedCommand class method), 338
cleanup() (datalad.utils.SwallowLogsAdapter method), 344
cleanup() (datalad.utils.SwallowOutputsAdapter method), 344
clear() (datalad.support.annexrepo.BatchedAnnexes method), 404
CLI, **141**
clone() (datalad.support.gitrepo.GitRepo class method), 370
clone() (in module datalad.api), 302
close() (datalad.cmd.BatchedCommand method), 338
close() (datalad.support.annexrepo.BatchedAnnexes method), 404
close_stdin() (datalad.runner.nonasyncrunner.ThreadedRunner method), 412
collect_method_callstats() (in module datalad.utils), 348
ColorFormatter (class in datalad.log), 340
commit() (datalad.support.gitrepo.GitRepo method), 371
commit_exists() (datalad.support.gitrepo.GitRepo method), 371
COMPRESSING (datalad.support.gitrepo.GitProgress attribute), 368
config (datalad.support.gitrepo.GitRepo property), 371
ConfigManager (class in datalad.config), 416
configuration() (in module datalad.api), 327
configure_fake_dates() (datalad.support.gitrepo.GitRepo method), 371
connection_lost() (datalad.runner.protocol.WitlessProtocol method), 414
connection_made() (datalad.runner.protocol.WitlessProtocol method), 414
connection_made() (datalad.support.annexrepo.AnnexJsonProtocol method), 387
connection_made() (datalad.support.gitrepo.GitProgress method), 368
copy_file() (in module datalad.api), 305
copy_to() (datalad.support.annexrepo.AnnexRepo method), 391
COST (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 409
COST (datalad.customremotes.base.AnnexCustomRemote attribute), 407
count_objects (datalad.support.gitrepo.GitRepo property), 371
COUNTING (datalad.support.gitrepo.GitProgress attribute), 368
create() (datalad.tests.utils_testrepos.InnerSubmodule method), 430
create() (datalad.tests.utils_testrepos.TestRepo method), 430
create() (in module datalad.api), 251
create_file() (datalad.tests.utils_testrepos.TestRepo method), 430
create_info_file() (datalad.tests.utils_testrepos.BasicAnnexTestRepo method), 430
create_info_file() (datalad.tests.utils_testrepos.BasicGitTestRepo method), 430
create_sibling() (in module datalad.api), 253
create_sibling_gin() (in module datalad.api), 266
create_sibling_gitea() (in module datalad.api), 264
create_sibling_github() (in module datalad.api), 256
create_sibling_gitlab() (in module datalad.api), 259
create_sibling_gogs() (in module datalad.api), 262
create_sibling_ria() (in module datalad.api), 268
create_test_dataset() (in module datalad.api), 307
create_tree() (in module datalad.utils), 348
create_tree_archive() (in module datalad.utils), 349
CUSTOM_REMOTE_NAME (datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute), 409

D

DataLad extension, **141**
datalad.annex.retry, **441**
datalad.cli.exec
    module, 435
datalad.cli.main
    module, 436
datalad.cli.parser
    module, 436
datalad.cli.renderer
    module, 438

datalad.clone.url-substitute.github, 439
datalad.clone.url-substitute.osf, 439
datalad.cmd
    module, 337
datalad.config
    module, 416
datalad.consts
    module, 340
datalad.crawl.cache, 440
datalad.credentials.force-ask, 441
datalad.credentials.githelper.noninteractive, 441
datalad.customremotes.archives
    module, 409
datalad.customremotes.base
    module, 407
datalad.exc.str.tblimit, 441
datalad.extensions.load, 439
datalad.externals.nda.dbserver, 439
datalad.fake-dates, 440
datalad.fake-dates-start, 441
datalad.github.token-note, 441
datalad.install.inherit-local-origin, 441
datalad.interface.base
    module, 431
datalad.locations.cache, 439
datalad.locations.dataset-procedures, 440
datalad.locations.default-dataset, 439
datalad.locations.extra-procedures, 439
datalad.locations.locks, 439
datalad.locations.sockets, 439
datalad.locations.system-procedures, 439
datalad.locations.user-procedures, 439
datalad.log
    module, 340
datalad.log.level, 441
datalad.log.name, 441
datalad.log.names, 441
datalad.log.namesre, 441
datalad.log.outputs, 441
datalad.log.result-level, 441
datalad.log.timestamp, 442
datalad.log.traceback, 442
datalad.repo.backend, 442
datalad.repo.direct, 442
datalad.repo.version, 442
datalad.runner.nonasyncrunner
    module, 411
datalad.runner.protocol
    module, 414
datalad.runtime.max-annex-jobs, 442
datalad.runtime.max-batched, 442
datalad.runtime.max-inactive-age, 442
datalad.runtime.max-jobs, 442
datalad.runtime.pathspec-from-file, 442
datalad.runtime.raiseonerror, 443
datalad.runtime.report-status, 443
datalad.runtime.stalled-external, 443
datalad.save.no-message, 443
datalad.save.windows-compat-warning, 443
datalad.source.epoch, 443
datalad.ssh.executable, 439
datalad.ssh.identityfile, 440
datalad.ssh.multiplex-connections, 440
datalad.ssh.try-use-annex-bundled-git, 440
datalad.support.annexrepo
    module, 387
datalad.support.archives
    module, 405
datalad.support.extensions
    module, 406
datalad.support.gitrepo
    module, 367
datalad.tests.cache, 440
datalad.tests.credentials, 440
datalad.tests.dataladremote, 443
datalad.tests.heavyoutput
    module, 431
datalad.tests.knownfailures.probe, 443
datalad.tests.knownfailures.skip, 443
datalad.tests.nonetwork, 443
datalad.tests.nonlo, 444
datalad.tests.noteardown, 444
datalad.tests.runcmdline, 444
datalad.tests.setup.testrepos, 444
datalad.tests.ssh, 444
datalad.tests.temp.dir, 444
datalad.tests.temp.fs, 444
datalad.tests.temp.fssize, 444
datalad.tests.temp.keep, 444
datalad.tests.ui.backend, 444
datalad.tests.usecassette, 444
datalad.tests.utils_pytest
    module, 422
datalad.tests.utils_testrepos
    module, 430
datalad.ui.color, 444
datalad.ui.progressbar, 444
datalad.ui.suppress-similar-results, 444
datalad.ui.suppress-similar-results-threshold, 445
datalad.utils
    module, 343
datalad.version
    module, 367
dataset, 141
Dataset (*class in datalad.api*), 249
decode_input() (*in module datalad.utils*), 349

decompress_file() (*in module datalad.support.archives*), 406
dedent_docstring() (*in module datalad.interface.base*), 433
default_backends (*datalad.support.annexrepo.AnnexRepo property*), 391
DefaultOutputFormatter (*class in datalad.cli.renderer*), 438
DefaultOutputRenderer (*class in datalad.cli.renderer*), 438
defaults (*datalad.utils.ArgSpecFake attribute*), 343
DELETED (*datalad.support.gitrepo.PushInfo attribute*), 385
describe() (*datalad.support.gitrepo.GitRepo method*), 371
diff() (*datalad.support.gitrepo.GitRepo method*), 372
diff() (*in module datalad.api*), 307
diffstatus() (*datalad.support.gitrepo.GitRepo method*), 372
dirty (*datalad.support.gitrepo.GitRepo property*), 372
disable_logger() (*in module datalad.utils*), 349
dlabspath() (*in module datalad.utils*), 349
DONE_TOKEN (*datalad.support.gitrepo.GitProgress attribute*), 368
download_url() (*in module datalad.api*), 309
drop() (*datalad.support.annexrepo.AnnexRepo method*), 391
drop() (*in module datalad.api*), 272
drop_key() (*datalad.support.annexrepo.AnnexRepo method*), 391

E

enable_remote() (*datalad.support.annexrepo.AnnexRepo method*), 391
encode_filename() (*in module datalad.utils*), 350
END (*datalad.support.gitrepo.GitProgress attribute*), 368
ensure_bool() (*in module datalad.utils*), 350
ensure_bytes() (*in module datalad.utils*), 350
ensure_datalad_remote() (*in module datalad.customremotes.base*), 408
ensure_dict_from_str() (*in module datalad.utils*), 350
ensure_dir() (*in module datalad.utils*), 350
ensure_iter() (*in module datalad.utils*), 350
ensure_list() (*in module datalad.utils*), 351
ensure_list_from_str() (*in module datalad.utils*), 351
ensure_result_list() (*in module datalad.utils*), 351
ensure_stdin_stdout_stderr_closed() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
ensure_stdout_stderr_closed() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
ensure_tuple_or_list() (*in module datalad.utils*), 351
ensure_unicode() (*in module datalad.utils*), 352
ensure_write_permission() (*in module datalad.utils*), 352
ENUMERATING (*datalad.support.gitrepo.GitProgress attribute*), 368
eq_() (*in module datalad.tests.utils_pytest*), 424
err (*datalad.utils.SwallowOutputsAdapter property*), 345
ERROR (*datalad.support.gitrepo.FetchInfo attribute*), 367
ERROR (*datalad.support.gitrepo.PushInfo attribute*), 385
escape_filename() (*in module datalad.utils*), 352
eval_results() (*in module datalad.interface.base*), 433
expandpath() (*in module datalad.utils*), 352
export_archive() (*in module datalad.api*), 329
export_archive_ora() (*in module datalad.api*), 331
export_to_figshare() (*in module datalad.api*), 332
ExtractedArchive (*class in datalad.support.archives*), 405

F

fail_with_short_help() (*in module datalad.cli.parser*), 436
fake_dates_enabled (*datalad.support.gitrepo.GitRepo property*), 373
FAST_FORWARD (*datalad.support.gitrepo.FetchInfo attribute*), 367
FAST_FORWARD (*datalad.support.gitrepo.PushInfo attribute*), 385
fd_infos (*datalad.support.annexrepo.AnnexInitOutput attribute*), 387
fd_infos (*datalad.support.annexrepo.AnnexJsonProtocol attribute*), 387
fd_infos (*datalad.support.annexrepo.GeneratorAnnexJsonNoStderrProto attribute*), 404
fd_infos (*datalad.support.annexrepo.GeneratorAnnexJsonProtocol attribute*), 404
fd_infos (*datalad.support.gitrepo.GitProgress attribute*), 368
fd_infos (*datalad.support.gitrepo.StdOutCaptureWithGitProgress attribute*), 385
fetch() (*datalad.support.gitrepo.GitRepo method*), 373
fetch_() (*datalad.support.gitrepo.GitRepo method*), 373
FetchInfo (*class in datalad.support.gitrepo*), 367
File (*class in datalad.utils*), 343
file (*datalad.support.gitrepo.GitAddOutput attribute*), 367
file_basename() (*in module datalad.utils*), 352

file_has_content() (*datalad.support.annexrepo.AnnexRepo method*), 391
filter_noninteractive_progress() (*in module datalad.log*), 341
find() (*datalad.support.annexrepo.AnnexRepo method*), 392
find_files() (*in module datalad.utils*), 353
FINDING_SOURCES (*datalad.support.gitrepo.GitProgress attribute*), 368
FORCED_UPDATE (*datalad.support.gitrepo.FetchInfo attribute*), 367
FORCED_UPDATE (*datalad.support.gitrepo.PushInfo attribute*), 385
foreach_dataset() (*in module datalad.api*), 311
format() (*datalad.log.ColorFormatter method*), 340
format_commit() (*datalad.support.gitrepo.GitRepo method*), 373
format_element() (*datalad.utils.SequenceFormatter method*), 343
format_field() (*datalad.utils.SequenceFormatter method*), 344
fsck() (*datalad.support.annexrepo.AnnexRepo method*), 392

G

gc() (*datalad.support.gitrepo.GitRepo method*), 373
gen_URLS() (*datalad.customremotes.base.AnnexCustomRemote method*), 407
generate_chunks() (*in module datalad.utils*), 353
generate_file_chunks() (*in module datalad.utils*), 353
generate_uuids() (*in module datalad.customremotes.base*), 409
GeneratorAnnexJsonNoStderrProtocol (*class in datalad.support.annexrepo*), 404
GeneratorAnnexJsonProtocol (*class in datalad.support.annexrepo*), 404
GeneratorMixIn (*class in datalad.runner.protocol*), 414
get() (*datalad.config.ConfigManager method*), 417
get() (*datalad.support.annexrepo.AnnexRepo method*), 392
get() (*datalad.support.annexrepo.BatchedAnnexes method*), 404
get() (*in module datalad.api*), 274
get_active_branch() (*datalad.support.gitrepo.GitRepo method*), 374
get_allargs_as_kwargs() (*in module datalad.interface.base*), 434
get_annexed_files() (*datalad.support.annexrepo.AnnexRepo method*), 393
get_annexstatus() (*in module datalad.tests.utils_pytest*), 424
get_api_name() (*in module datalad.interface.base*), 434
get_archive() (*datalad.support.archives.ArchivesCache method*), 405
get_branch_commits_() (*datalad.support.gitrepo.GitRepo method*), 374
get_branches() (*datalad.support.gitrepo.GitRepo method*), 374
get_cmd_doc() (*in module datalad.interface.base*), 434
get_cmd_summaries() (*in module datalad.interface.base*), 434
get_commit_date() (*datalad.support.gitrepo.GitRepo method*), 374
get_content_annexinfo() (*datalad.support.annexrepo.AnnexRepo method*), 393
get_content_info() (*datalad.support.gitrepo.GitRepo method*), 374
get_contentlocation() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 410
get_contentlocation() (*datalad.support.annexrepo.AnnexRepo method*), 393
get_convoluted_situation() (*in module datalad.tests.utils_pytest*), 424
get_corresponding_branch() (*datalad.support.annexrepo.AnnexRepo method*), 394
get_corresponding_branch() (*datalad.support.gitrepo.GitRepo method*), 375
get_dataset_root() (*in module datalad.utils*), 353
get_datasets_topdir() (*in module datalad.tests.utils_pytest*), 424
get_deeply_nested_structure() (*in module datalad.tests.utils_pytest*), 424
get_description() (*datalad.support.annexrepo.AnnexRepo method*), 394
get_encoding_info() (*in module datalad.utils*), 353
get_envvars_info() (*in module datalad.utils*), 354
get_extracted_file() (*datalad.support.archives.ExtractedArchive method*), 405
get_extracted_filename() (*datalad.support.archives.ExtractedArchive method*), 405
get_extracted_files() (*datalad.support.archives.ExtractedArchive method*), 405
get_file_annexinfo() (*datalad.support.annexrepo.AnnexRepo method*), 394

get_file_backend() (*datalad.support.annexrepo.AnnexRepo method*), 395
get_file_key() (*datalad.support.annexrepo.AnnexRepo method*), 395
get_file_size() (*datalad.support.annexrepo.AnnexRepo method*), 395
get_file_url() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 410
get_files() (*datalad.support.gitrepo.GitRepo method*), 375
get_from_source() (*datalad.config.ConfigManager method*), 417
get_git_attributes() (*datalad.support.gitrepo.GitRepo method*), 376
get_git_dir() (*datalad.support.gitrepo.GitRepo static method*), 376
get_git_version() (*in module datalad.config*), 420
get_gitattributes() (*datalad.support.gitrepo.GitRepo method*), 376
get_groupwanted() (*datalad.support.annexrepo.AnnexRepo method*), 395
get_hexsha() (*datalad.support.gitrepo.GitRepo method*), 376
get_home_envvars() (*in module datalad.utils*), 354
get_indexed_files() (*datalad.support.gitrepo.GitRepo method*), 377
get_initialized_logger() (*datalad.log.LoggerHelper method*), 340
get_interface_groups() (*in module datalad.interface.base*), 434
get_ipython_shell() (*in module datalad.utils*), 354
get_key_backend() (*datalad.support.annexrepo.AnnexRepo class method*), 395
get_last_commit_hexsha() (*datalad.support.gitrepo.GitRepo method*), 377
get_leading_directory() (*datalad.support.archives.ExtractedArchive method*), 405
get_linux_distribution() (*in module datalad.utils*), 354
get_logfilename() (*in module datalad.utils*), 354
get_merge_base() (*datalad.support.gitrepo.GitRepo method*), 377
get_metadata() (*datalad.support.annexrepo.AnnexRepo method*), 395
get_most_obscure_supported_name() (*in module datalad.tests.utils_pytest*), 425
get_mtimes_and_digests() (*in module datalad.tests.utils_pytest*), 425
get_one_line() (*datalad.cmd.BatchedCommand method*), 338
get_open_files() (*in module datalad.utils*), 354
get_path_prefix() (*in module datalad.utils*), 355
get_preferred_content() (*datalad.support.annexrepo.AnnexRepo method*), 396
get_refds_path() (*datalad.interface.base.Interface class method*), 433
get_remote_branches() (*datalad.support.gitrepo.GitRepo method*), 377
get_remote_url() (*datalad.support.gitrepo.GitRepo method*), 377
get_remotes() (*datalad.support.annexrepo.AnnexRepo method*), 396
get_remotes() (*datalad.support.gitrepo.GitRepo method*), 377
get_requested_error_output() (*datalad.cmd.BatchedCommand method*), 338
get_revisions() (*datalad.support.gitrepo.GitRepo method*), 378
get_sig_param_names() (*in module datalad.utils*), 355
get_size_from_key() (*datalad.support.annexrepo.AnnexRepo static method*), 396
get_special_remotes() (*datalad.support.annexrepo.AnnexRepo method*), 396
get_ssh_port() (*in module datalad.tests.utils_pytest*), 425
get_staged_paths() (*datalad.support.gitrepo.GitRepo method*), 378
get_submodules() (*datalad.support.gitrepo.GitRepo method*), 378
get_submodules_() (*datalad.support.gitrepo.GitRepo method*), 378
get_suggestions_msg() (*in module datalad.utils*), 355
get_tags() (*datalad.support.gitrepo.GitRepo method*), 379
get_tempfile_kwargs() (*in module datalad.utils*), 355
get_timeout_exception() (*datalad.cmd.BatchedCommand method*), 338
get_timestamp_suffix() (*in module datalad.utils*), 356
get_toppath() (*datalad.support.gitrepo.GitRepo class method*), 379
get_trace() (*in module datalad.utils*), 356
get_tracking_branch() (*datalad.support.annexrepo.AnnexRepo method*), 397
get_tracking_branch() (*datalad.support.gitrepo.GitRepo method*), 379

get_urls() (*datalad.support.annexrepo.AnnexRepo method*), 397
get_value() (*datalad.cli.renderer.DefaultOutputFormatter method*), 438
get_value() (*datalad.config.ConfigManager method*), 417
get_wrapped_class() (*in module datalad.utils*), 356
getargspec() (*in module datalad.utils*), 356
getavailability() (*datalad.customremotes.base.AnnexCustomRemote method*), 407
getbool() (*datalad.config.ConfigManager method*), 417
getcost() (*datalad.customremotes.base.AnnexCustomRemote method*), 407
getfloat() (*datalad.config.ConfigManager method*), 417
getint() (*datalad.config.ConfigManager method*), 417
getpwd() (*in module datalad.utils*), 357
GIT_ANNEX_MIN_VERSION (*datalad.support.annexrepo.AnnexRepo attribute*), 387
git_annex_version (*datalad.support.annexrepo.AnnexRepo attribute*), 397
GIT_MIN_VERSION (*datalad.support.gitrepo.GitRepo attribute*), 369
git_version (*datalad.support.gitrepo.GitRepo attribute*), 379
GitAddOutput (*class in datalad.support.gitrepo*), 367
GitProgress (*class in datalad.support.gitrepo*), 367
GitRepo (*class in datalad.support.gitrepo*), 368
guard_for_format() (*in module datalad.utils*), 357

H

handle (*datalad.utils.SwallowLogsAdapter property*), 344
handles (*datalad.utils.SwallowOutputsAdapter property*), 345
has_config() (*in module datalad.support.extensions*), 406
has_option() (*datalad.config.ConfigManager method*), 417
has_section() (*datalad.config.ConfigManager method*), 417
has_symlink_capability() (*in module datalad.tests.utils_pytest*), 425
HEAD_UPTODATE (*datalad.support.gitrepo.FetchInfo attribute*), 367
HTTPPath (*class in datalad.tests.utils_pytest*), 422

I

ignore_nose_capturing_stdout() (*in module datalad.tests.utils_pytest*), 425
import_module_from_file() (*in module datalad.utils*), 357
import_modules() (*in module datalad.utils*), 357
in_() (*in module datalad.tests.utils_pytest*), 425
info() (*datalad.support.annexrepo.AnnexRepo method*), 397
init_datalad_remote() (*in module datalad.customremotes.base*), 409
init_remote() (*datalad.support.annexrepo.AnnexRepo method*), 397
initremote() (*datalad.customremotes.base.AnnexCustomRemote method*), 408
InnerSubmodule (*class in datalad.tests.utils_testrepos*), 430
install() (*in module datalad.api*), 277
integration() (*in module datalad.tests.utils_pytest*), 425
Interface (*class in datalad.interface.base*), 431
is_ancestor() (*datalad.support.gitrepo.GitRepo method*), 380
is_api_arg() (*in module datalad.interface.base*), 435
is_available() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_crippled_fs() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_direct_mode() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_explicit_path() (*in module datalad.utils*), 358
is_extracted (*datalad.support.archives.ExtractedArchive property*), 406
is_initialized() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_interactive() (*in module datalad.utils*), 358
is_managed_branch() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_remote_annex_ignored() (*datalad.support.annexrepo.AnnexRepo method*), 398
is_special_annex_remote() (*datalad.support.annexrepo.AnnexRepo method*), 399
is_stalled() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
is_under_annex() (*datalad.support.annexrepo.AnnexRepo method*), 399
is_valid_annex() (*datalad.support.annexrepo.AnnexRepo method*), 399

is_valid_git() (*datalad.support.gitrepo.GitRepo method*), 380
is_valid_repo() (*datalad.support.annexrepo.AnnexRepo class method*), 399
is_valid_repo() (*datalad.support.gitrepo.GitRepo class method*), 380
is_with_annex() (*datalad.support.gitrepo.GitRepo method*), 380
items() (*datalad.config.ConfigManager method*), 417

J

join_cmdline() (*in module datalad.utils*), 358

K

keys() (*datalad.config.ConfigManager method*), 417
keywords (*datalad.utils.ArgSpecFake attribute*), 343
known_failure() (*in module datalad.tests.utils_pytest*), 425
known_failure_direct_mode() (*in module datalad.tests.utils_pytest*), 425
known_failure_githubci_osx() (*in module datalad.tests.utils_pytest*), 426
known_failure_githubci_win() (*in module datalad.tests.utils_pytest*), 426
known_failure_osx() (*in module datalad.tests.utils_pytest*), 426
known_failure_windows() (*in module datalad.tests.utils_pytest*), 426
knows_annex() (*in module datalad.utils*), 358

L

line_profile() (*in module datalad.utils*), 358
lines (*datalad.utils.SwallowLogsAdapter property*), 344
link_file_load() (*in module datalad.customremotes.archives*), 411
lmtime() (*in module datalad.utils*), 358
load_interface() (*in module datalad.interface.base*), 435
localsync() (*datalad.support.annexrepo.AnnexRepo method*), 399
lock_if_required() (*in module datalad.utils*), 359
log_message() (*datalad.tests.utils_pytest.SilentHTTPHandler method*), 422
log_progress() (*in module datalad.log*), 341
LoggerHelper (*class in datalad.log*), 340

M

main() (*in module datalad.cli.main*), 436
main() (*in module datalad.customremotes.archives*), 411
make_tempfile() (*in module datalad.utils*), 359
map_items() (*in module datalad.utils*), 360
maybe_adjust_repo() (*in module datalad.tests.utils_pytest*), 426
md5sum() (*in module datalad.utils*), 360
merge() (*datalad.support.gitrepo.GitRepo method*), 380
merge_annex() (*datalad.support.annexrepo.AnnexRepo method*), 399
migrate_backend() (*datalad.support.annexrepo.AnnexRepo method*), 399
module
- datalad.cli.exec, 435
- datalad.cli.main, 436
- datalad.cli.parser, 436
- datalad.cli.renderer, 438
- datalad.cmd, 337
- datalad.config, 416
- datalad.consts, 340
- datalad.customremotes.archives, 409
- datalad.customremotes.base, 407
- datalad.interface.base, 431
- datalad.log, 340
- datalad.runner.nonasyncrunner, 411
- datalad.runner.protocol, 414
- datalad.support.annexrepo, 387
- datalad.support.archives, 405
- datalad.support.extensions, 406
- datalad.support.gitrepo, 367
- datalad.tests.heavyoutput, 431
- datalad.tests.utils_pytest, 422
- datalad.tests.utils_testrepos, 430
- datalad.utils, 343
- datalad.version, 367

N

nadict() (*in module datalad.cli.renderer*), 438
nagen (*class in datalad.cli.renderer*), 438
neq_() (*in module datalad.tests.utils_pytest*), 426
NestedDataset (*class in datalad.tests.utils_testrepos*), 430
never_fail() (*in module datalad.utils*), 360
NEW_HEAD (*datalad.support.gitrepo.FetchInfo attribute*), 367
NEW_HEAD (*datalad.support.gitrepo.PushInfo attribute*), 385
NEW_TAG (*datalad.support.gitrepo.FetchInfo attribute*), 367
NEW_TAG (*datalad.support.gitrepo.PushInfo attribute*), 385
no_annex() (*in module datalad.api*), 333
NO_MATCH (*datalad.support.gitrepo.PushInfo attribute*), 385
nok_() (*in module datalad.tests.utils_pytest*), 426
nok_startswith() (*in module datalad.tests.utils_pytest*), 426

normalize_path() (*in module datalad.support.gitrepo*), 385
normalize_paths() (*in module datalad.support.gitrepo*), 386
not_supported_on_windows() (*in module datalad.utils*), 360
nothing_cm() (*in module datalad.utils*), 360

O

obtain() (*datalad.config.ConfigManager method*), 418
obtain_write_permission() (*in module datalad.utils*), 360
ok_() (*in module datalad.tests.utils_pytest*), 426
ok_annex_get() (*in module datalad.tests.utils_pytest*), 426
ok_archives_caches() (*in module datalad.tests.utils_pytest*), 426
ok_broken_symlink() (*in module datalad.tests.utils_pytest*), 426
ok_clean_git() (*in module datalad.tests.utils_pytest*), 426
ok_endswith() (*in module datalad.tests.utils_pytest*), 426
ok_exists() (*in module datalad.tests.utils_pytest*), 426
ok_file_has_content() (*in module datalad.tests.utils_pytest*), 426
ok_file_under_git() (*in module datalad.tests.utils_pytest*), 426
ok_generator() (*in module datalad.tests.utils_pytest*), 427
ok_git_config_not_empty() (*in module datalad.tests.utils_pytest*), 427
ok_good_symlink() (*in module datalad.tests.utils_pytest*), 427
ok_startswith() (*in module datalad.tests.utils_pytest*), 427
ok_symlink() (*in module datalad.tests.utils_pytest*), 427
on_failure (*datalad.interface.base.Interface attribute*), 433
OP_MASK (*datalad.support.gitrepo.GitProgress attribute*), 368
open_r_encdetect() (*in module datalad.utils*), 361
optional_args() (*in module datalad.utils*), 361
options() (*datalad.config.ConfigManager method*), 418
out (*datalad.utils.SwallowLogsAdapter property*), 344
out (*datalad.utils.SwallowOutputsAdapter property*), 345

P

parse_gitconfig_dump() (*in module datalad.config*), 420
parser_add_common_opt() (*in module datalad.cli.parser*), 436
parser_add_common_options() (*in module datalad.cli.parser*), 436
parser_add_version_opt() (*in module datalad.cli.parser*), 436
partition() (*in module datalad.utils*), 361
patch_config() (*in module datalad.tests.utils_pytest*), 427
path (*datalad.support.archives.ArchivesCache property*), 405
path (*datalad.support.archives.ExtractedArchive property*), 406
path (*datalad.tests.utils_testrepos.InnerSubmodule property*), 430
path (*datalad.tests.utils_testrepos.TestRepo property*), 430
path_is_subpath() (*in module datalad.utils*), 362
path_startswith() (*in module datalad.utils*), 362
pipe_connection_lost() (*datalad.cmd.BatchedCommandProtocol method*), 339
pipe_connection_lost() (*datalad.runner.protocol.WitlessProtocol method*), 414
pipe_data_received() (*datalad.cmd.BatchedCommandProtocol method*), 339
pipe_data_received() (*datalad.runner.protocol.WitlessProtocol method*), 415
pipe_data_received() (*datalad.support.annexrepo.AnnexInitOutput method*), 387
pipe_data_received() (*datalad.support.annexrepo.AnnexJsonProtocol method*), 387
pipe_data_received() (*datalad.support.annexrepo.GeneratorAnnexJsonNoStderrProtocol method*), 404
pipe_data_received() (*datalad.support.gitrepo.GitProgress method*), 368
populate() (*datalad.tests.utils_testrepos.BasicAnnexTestRepo method*), 430
populate() (*datalad.tests.utils_testrepos.BasicGitTestRepo method*), 430
populate() (*datalad.tests.utils_testrepos.NestedDataset method*), 430
populate() (*datalad.tests.utils_testrepos.SubmoduleDataset method*), 430
populate() (*datalad.tests.utils_testrepos.TestRepo method*), 430
posix_relpath() (*in module datalad.utils*), 362
precommit() (*datalad.support.annexrepo.AnnexRepo method*), 400

precommit() (*datalad.support.gitrepo.GitRepo method*), 380
prepare() (*datalad.customremotes.base.AnnexCustomRemote method*), 408
probe_known_failure() (*in module datalad.tests.utils_pytest*), 427
proc1() (*datalad.cmd.BatchedCommand method*), 338
proc_err (*datalad.runner.protocol.WitlessProtocol attribute*), 415
proc_err (*datalad.support.annexrepo.AnnexInitOutput attribute*), 387
proc_err (*datalad.support.annexrepo.AnnexJsonProtocol attribute*), 387
proc_err (*datalad.support.gitrepo.GitProgress attribute*), 368
proc_out (*datalad.runner.protocol.WitlessProtocol attribute*), 415
proc_out (*datalad.support.annexrepo.AnnexInitOutput attribute*), 387
proc_out (*datalad.support.annexrepo.AnnexJsonProtocol attribute*), 387
proc_out (*datalad.support.gitrepo.StdOutCaptureWithGitProgress attribute*), 385
process (*datalad.support.annexrepo.AnnexInitOutput attribute*), 387
process (*datalad.support.annexrepo.AnnexJsonProtocol attribute*), 387
process (*datalad.support.annexrepo.GeneratorAnnexJsonNoStderrProtocol attribute*), 404
process (*datalad.support.annexrepo.GeneratorAnnexJsonProtocol attribute*), 404
process (*datalad.support.gitrepo.GitProgress attribute*), 368
process (*datalad.support.gitrepo.StdOutCaptureWithGitProgress attribute*), 385
process_exited() (*datalad.runner.protocol.WitlessProtocol method*), 415
process_exited() (*datalad.support.annexrepo.AnnexJsonProtocol method*), 387
process_exited() (*datalad.support.annexrepo.GeneratorAnnexJsonNoStderrProtocol method*), 404
process_exited() (*datalad.support.gitrepo.GitProgress method*), 368
process_loop() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
process_queue() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
process_request() (*datalad.cmd.BatchedCommand method*), 338
process_running() (*datalad.cmd.BatchedCommand method*), 339
process_timeouts() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
push() (*datalad.support.gitrepo.GitRepo method*), 381
push() (*in module datalad.api*), 279
push_() (*datalad.support.gitrepo.GitRepo method*), 381
PushInfo (*class in datalad.support.gitrepo*), 385
put_file_under_git() (*in module datalad.tests.utils_pytest*), 427

Q

quote_cmdlinearg() (*in module datalad.utils*), 362
quote_config() (*in module datalad.config*), 421

R

re_op_absolute (*datalad.support.gitrepo.GitProgress attribute*), 368
re_op_relative (*datalad.support.gitrepo.GitProgress attribute*), 368
read_csv_lines() (*in module datalad.utils*), 362
read_file() (*in module datalad.utils*), 363
readline() (*datalad.cmd.ReadlineEmulator method*), 340
readline_json() (*in module datalad.support.annexrepo*), 404
readline_rstripped() (*in module datalad.cmd*), 340
ReadlineEmulator (*class in datalad.cmd*), 340
readlines_until_ok_or_failed() (*in module datalad.support.annexrepo*), 404
RECEIVING (*datalad.support.gitrepo.GitProgress attribute*), 368
register_config() (*in module datalad.support.extensions*), 406
REJECTED (*datalad.support.gitrepo.FetchInfo attribute*), 367
REJECTED (*datalad.support.gitrepo.PushInfo attribute*), 385
reload() (*datalad.config.ConfigManager method*), 418
REMOTE_FAILURE (*datalad.support.gitrepo.PushInfo attribute*), 385
REMOTE_REJECTED (*datalad.support.gitrepo.PushInfo attribute*), 385
remove() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 410
remove() (*datalad.customremotes.base.AnnexCustomRemote method*), 408
remove() (*datalad.support.gitrepo.GitRepo method*), 381
remove() (*in module datalad.api*), 281
remove_branch() (*datalad.support.gitrepo.GitRepo method*), 382

remove_file_number() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
remove_process() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
remove_remote() (*datalad.support.gitrepo.GitRepo method*), 382
remove_section() (*datalad.config.ConfigManager method*), 418
rename_section() (*datalad.config.ConfigManager method*), 419
REPO_CLASS (*datalad.tests.utils_testrepos.BasicAnnexTestRepo attribute*), 430
REPO_CLASS (*datalad.tests.utils_testrepos.BasicGitTestRepo attribute*), 430
REPO_CLASS (*datalad.tests.utils_testrepos.TestRepo attribute*), 430
repo_info() (*datalad.support.annexrepo.AnnexRepo method*), 400
repository_versions (*datalad.support.annexrepo.AnnexRepo attribute*), 400
rerun() (*in module datalad.api*), 295
RESOLVING (*datalad.support.gitrepo.GitProgress attribute*), 368
result_filter (*datalad.interface.base.Interface attribute*), 433
result_renderer (*datalad.interface.base.Interface attribute*), 433
result_xfm (*datalad.interface.base.Interface attribute*), 433
return_type (*datalad.interface.base.Interface attribute*), 433
rewrite_url() (*datalad.config.ConfigManager method*), 419
rewrite_url() (*in module datalad.config*), 421
rm_url() (*datalad.support.annexrepo.AnnexRepo method*), 400
rmdir() (*in module datalad.utils*), 363
rmtemp() (*in module datalad.utils*), 363
rmtree() (*in module datalad.utils*), 363
rotree() (*in module datalad.utils*), 364
run() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 412
run() (*in module datalad.api*), 292
run_command() (*in module datalad.runner.nonasyncrunner*), 413
run_procedure() (*in module datalad.api*), 298
run_under_dir() (*in module datalad.tests.utils_pytest*), 427

S

SafeDelCloseMixin (*class in datalad.cmd*), 340
save() (*datalad.support.gitrepo.GitRepo method*), 382
save() (*in module datalad.api*), 283
save_() (*datalad.support.gitrepo.GitRepo method*), 382
saved_generator() (*in module datalad.utils*), 364
sections() (*datalad.config.ConfigManager method*), 419
send_result() (*datalad.runner.protocol.GeneratorMixIn method*), 414
SequenceFormatter (*class in datalad.utils*), 343
serve_path_via_http() (*in module datalad.tests.utils_pytest*), 427
set() (*datalad.config.ConfigManager method*), 419
set_annex_version() (*in module datalad.tests.utils_pytest*), 427
set_date() (*in module datalad.tests.utils_pytest*), 427
set_default_backend() (*datalad.support.annexrepo.AnnexRepo method*), 400
set_gitattributes() (*datalad.support.gitrepo.GitRepo method*), 383
set_groupwanted() (*datalad.support.annexrepo.AnnexRepo method*), 400
set_level() (*datalad.log.LoggerHelper method*), 341
set_metadata() (*datalad.support.annexrepo.AnnexRepo method*), 400
set_metadata_() (*datalad.support.annexrepo.AnnexRepo method*), 401
set_preferred_content() (*datalad.support.annexrepo.AnnexRepo method*), 401
set_remote_dead() (*datalad.support.annexrepo.AnnexRepo method*), 401
set_remote_url() (*datalad.support.annexrepo.AnnexRepo method*), 401
set_remote_url() (*datalad.support.gitrepo.GitRepo method*), 383
setup_parser() (*in module datalad.cli.parser*), 437
setup_parser_for_interface() (*in module datalad.cli.parser*), 437
setup_parserarg_for_interface() (*in module datalad.cli.parser*), 437
shell_completion() (*in module datalad.api*), 335
shortened_repr() (*in module datalad.utils*), 364
should_continue() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 413
sibling, **141**
siblings() (*in module datalad.api*), 313

SilentHTTPHandler (*class in datalad.tests.utils_pytest*), 422
single_subparser_possible() (*in module datalad.cli.parser*), 437
skip_if() (*in module datalad.tests.utils_pytest*), 428
skip_if_adjusted_branch() (*in module datalad.tests.utils_pytest*), 428
skip_if_no_module() (*in module datalad.tests.utils_pytest*), 428
skip_if_no_network() (*in module datalad.tests.utils_pytest*), 428
skip_if_on_windows() (*in module datalad.tests.utils_pytest*), 428
skip_if_root() (*in module datalad.tests.utils_pytest*), 428
skip_if_scrapy_without_selector() (*in module datalad.tests.utils_pytest*), 428
skip_if_url_is_not_available() (*in module datalad.tests.utils_pytest*), 428
skip_known_failure() (*in module datalad.tests.utils_pytest*), 428
skip_nomultiplex_ssh() (*in module datalad.tests.utils_pytest*), 428
skip_ssh() (*in module datalad.tests.utils_pytest*), 428
skip_wo_symlink_capability() (*in module datalad.tests.utils_pytest*), 428
slash_join() (*in module datalad.utils*), 364
slow() (*in module datalad.tests.utils_pytest*), 428
split_cmdline() (*in module datalad.utils*), 364
sshrun() (*in module datalad.api*), 316
STAGE_MASK (*datalad.support.gitrepo.GitProgress attribute*), 368
stamp_path (*datalad.support.archives.ExtractedArchive property*), 406
STAMP_SUFFIX (*datalad.support.archives.ExtractedArchive attribute*), 405
start() (*datalad.tests.utils_pytest.HTTPPath method*), 422
status() (*datalad.support.gitrepo.GitRepo method*), 383
status() (*in module datalad.api*), 286
StdOutCaptureWithGitProgress (*class in datalad.support.gitrepo*), 385
stop() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 410
stop() (*datalad.tests.utils_pytest.HTTPPath method*), 422
subdataset, **141**
subdatasets() (*in module datalad.api*), 316
SubmoduleDataset (*class in datalad.tests.utils_testrepos*), 430
success (*datalad.support.gitrepo.GitAddOutput attribute*), 367
superdataset, **141**
SUPPORTED_SCHEMES (*datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute*), 409
supports_direct_mode (*datalad.support.annexrepo.AnnexRepo attribute*), 402
supports_unlocked_pointers (*datalad.support.annexrepo.AnnexRepo property*), 402
swallow_logs() (*in module datalad.utils*), 365
swallow_outputs() (*in module datalad.utils*), 365
SwallowLogsAdapter (*class in datalad.utils*), 344
SwallowOutputsAdapter (*class in datalad.utils*), 344
sync() (*datalad.support.annexrepo.AnnexRepo method*), 402

T

tag() (*datalad.support.gitrepo.GitRepo method*), 384
TAG_UPDATE (*datalad.support.gitrepo.FetchInfo attribute*), 367
TestRepo (*class in datalad.tests.utils_testrepos*), 430
ThreadedRunner (*class in datalad.runner.nonasyncrunner*), 411
timeout() (*datalad.cmd.BatchedCommandProtocol method*), 339
timeout() (*datalad.runner.protocol.WitlessProtocol method*), 415
timeout_resolution (*datalad.runner.nonasyncrunner.ThreadedRunner attribute*), 413
to_options() (*in module datalad.support.gitrepo*), 386
todo_interface_for_extensions() (*in module datalad.utils*), 365
TOKEN_SEPARATOR (*datalad.support.gitrepo.GitProgress attribute*), 368
transfer_retrieve() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 411
transfer_store() (*datalad.customremotes.base.AnnexCustomRemote method*), 408
try_multiple() (*in module datalad.utils*), 365
try_multiple_dec() (*in module datalad.utils*), 365
try_suggest_extension_with_command() (*in module datalad.cli.parser*), 437
turtle() (*in module datalad.tests.utils_pytest*), 429

U

unannex() (*datalad.support.annexrepo.AnnexRepo method*), 402
unique() (*in module datalad.utils*), 366
unlink() (*in module datalad.utils*), 366
unlock() (*datalad.support.annexrepo.AnnexRepo method*), 402

unlock() (*in module datalad.api*), 291
unset() (*datalad.config.ConfigManager method*), 420
untracked_files (*datalad.support.gitrepo.GitRepo property*), 384
UP_TO_DATE (*datalad.support.gitrepo.PushInfo attribute*), 385
update() (*in module datalad.api*), 289
update_docstring_with_examples() (*in module datalad.interface.base*), 435
update_docstring_with_parameters() (*in module datalad.interface.base*), 435
update_ref() (*datalad.support.gitrepo.GitRepo method*), 384
update_remote() (*datalad.support.gitrepo.GitRepo method*), 384
updated() (*in module datalad.utils*), 366
url (*datalad.tests.utils_testrepos.InnerSubmodule property*), 430
url (*datalad.tests.utils_testrepos.TestRepo property*), 431
URL_PREFIX (*datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute*), 409
URL_SCHEME (*datalad.customremotes.archives.ArchiveAnnexCustomRemote attribute*), 409
usecase() (*in module datalad.tests.utils_pytest*), 429
uuid (*datalad.support.annexrepo.AnnexRepo property*), 403

V

vararg s (*datalad.utils.ArgSpecFake attribute*), 343

W

wait_for_threads() (*datalad.runner.nonasyncrunner.ThreadedRunner method*), 413
warn_on_undefined_git_identity() (*in module datalad.config*), 421
WEB_UUID (*datalad.support.annexrepo.AnnexRepo attribute*), 387
whereis() (*datalad.customremotes.archives.ArchiveAnnexCustomRemote method*), 411
whereis() (*datalad.support.annexrepo.AnnexRepo method*), 403
with_fake_cookies_db() (*in module datalad.tests.utils_pytest*), 429
with_memory_keyring() (*in module datalad.tests.utils_pytest*), 429
with_pathsep() (*in module datalad.utils*), 366
with_progress() (*in module datalad.log*), 342
with_result_progress() (*in module datalad.log*), 342
with_sameas_remote() (*in module datalad.tests.utils_pytest*), 429
with_tempfile() (*in module datalad.tests.utils_pytest*), 429
with_testsui() (*in module datalad.tests.utils_pytest*), 429
with_tree() (*in module datalad.tests.utils_pytest*), 429
without_http_proxy() (*in module datalad.tests.utils_pytest*), 429
WitlessProtocol (*class in datalad.runner.protocol*), 414
write_config_section() (*in module datalad.config*), 421
WRITING (*datalad.support.gitrepo.GitProgress attribute*), 368
wtf() (*in module datalad.api*), 336