# Datalad Gooey

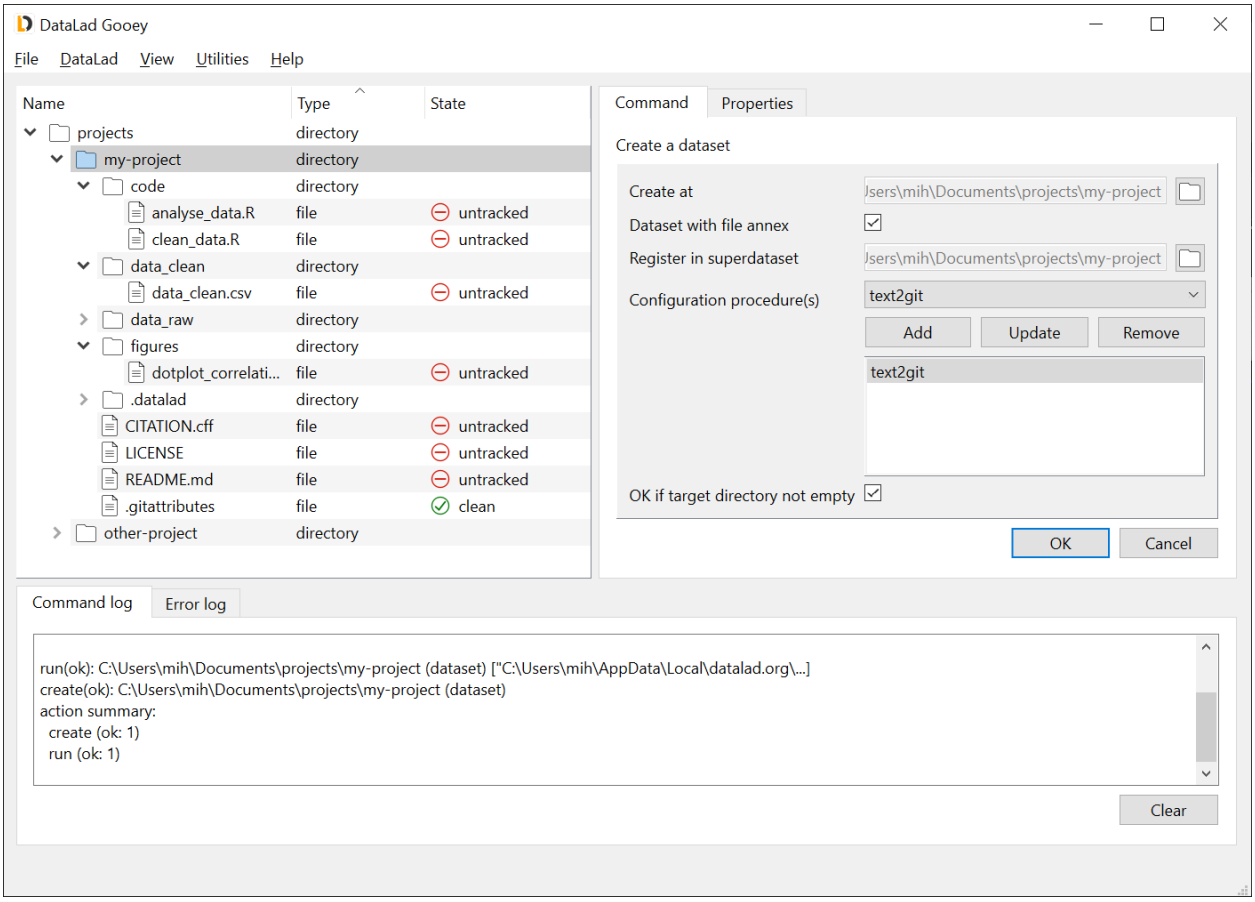
***Release 0.2.0+43.g5bd6b92***

**DataLad team**

**Feb 20, 2024**

# CONTENTS

DataLad Gooey is a Graphical User Interface (GUI) for using DataLad, a free and open source distributed data management tool. DataLad Gooey is compatible with all major operating systems and allows access to DataLad's operations via both a simplified and complete suite.




While using DataLad Gooey assumes at least some familiarity with DataLad concepts, the simplified command suite makes starting with DataLad easier via tailor-made command selections, condensed parameter specifications, and tool tips. The current core functionality supported via the simplified suite includes:

- cloning a dataset
- creating a dataset
- creating a sibling (GIN, GitHub, WebDAV)
- dropping/getting content
- pushing data/updates to a sibling
- saving the state of a dataset
- updating from a sibling

In addition, DataLad Gooey adds support for querying and setting *credentials*, *git-annex metadata*, and *general metadata*.

CHAPTER

ONE

# WHAT DATALAD GOOEY IS NOT

DataLad Gooey has a number of cool features, but here are features that you will need to use other tools for:

- An interface to visualize revision histories of DataLad datasets. Please refer to many of the available visual Git visualization tools
- An interface for advanced Git operations such as branching, resetting, reverting, or otherwise interacting with commit history. Please refer to your favourite Git client or the command line for these operations. DataLad Gooey will detect such external operations, and will update its view accordingly.

CHAPTER

TWO

# OVERVIEW

## 2.1 Installation

### 2.1.1 Installing via PyPI

You can install the latest version of `datalad-gooey` from PyPI. It is recommended to use a dedicated virtualenv:

```
# Create and enter a new virtual environment (optional)
python3 -m venv ~/.venvs/datalad-gooey
source ~/.venvs/datalad-gooey/bin/activate
```

```
# Install from PyPI
pip install datalad_gooey
```

**Dependencies**

Because this is an extension to `datalad`, the installation process also installs the datalad Python package, although all recursive dependencies (such as `git-annex`) are not automatically installed. For complete instructions on how to install `datalad` and `git-annex`, please refer to the DataLad Handbook.

### 2.1.2 Installing on Windows

The current version of `datalad-gooey` comes with two installers, a full installer and a gooey-only installer. The full installer will install `datalad-gooey` as well as `git` and `git-annex`. it requires admin privileges to execute successfully. The gooey-only installer will only install `datalad-gooey`. It can be executed as admin user or as non-admin user. If you use the gooey-only installer, `git` and `git-annex` have to be provided by other means, e.g. an administrator installs them.

Note: if the full installer is executed and the system has already a newer version of `git` or `git-annex` installed, the `git` or `git-annex`-installer can be canceled and installation of the remaining components will continue.

The installers can be downloaded here.

### 2.1.3 Installing on Linux

Install DataLad Gooey via PyPI while specifying the `--user` flag:

```
pip install --user datalad_gooey
```

Then run the following line to ensure that the application's desktop file is generated:

```
datalad gooey --postinstall
```

### 2.1.4 Installing on macOS

**macOS application pending**

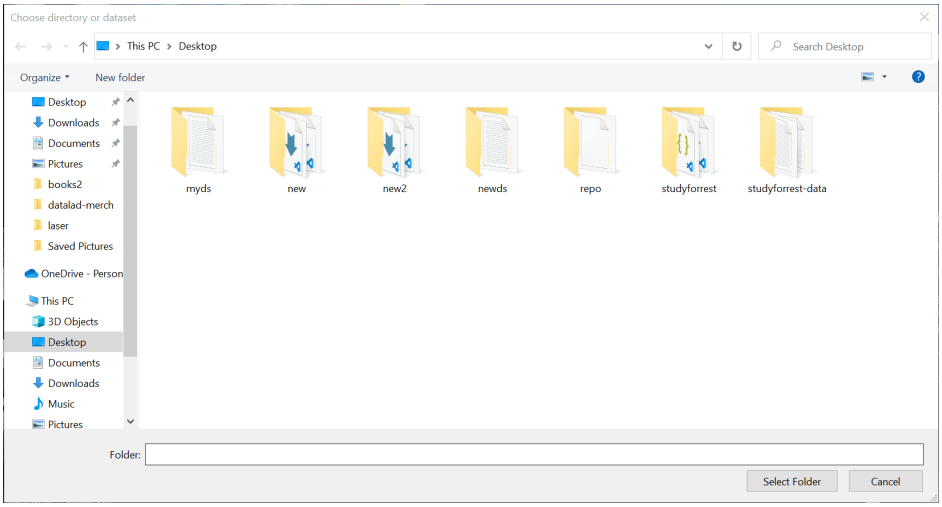
Until the macOS application is available to allow standard installation into the Applications folder, macOS users can install DataLad Gooey via PyPI.

```
# Install from PyPI
pip install datalad_gooey
```

## 2.2 Getting started

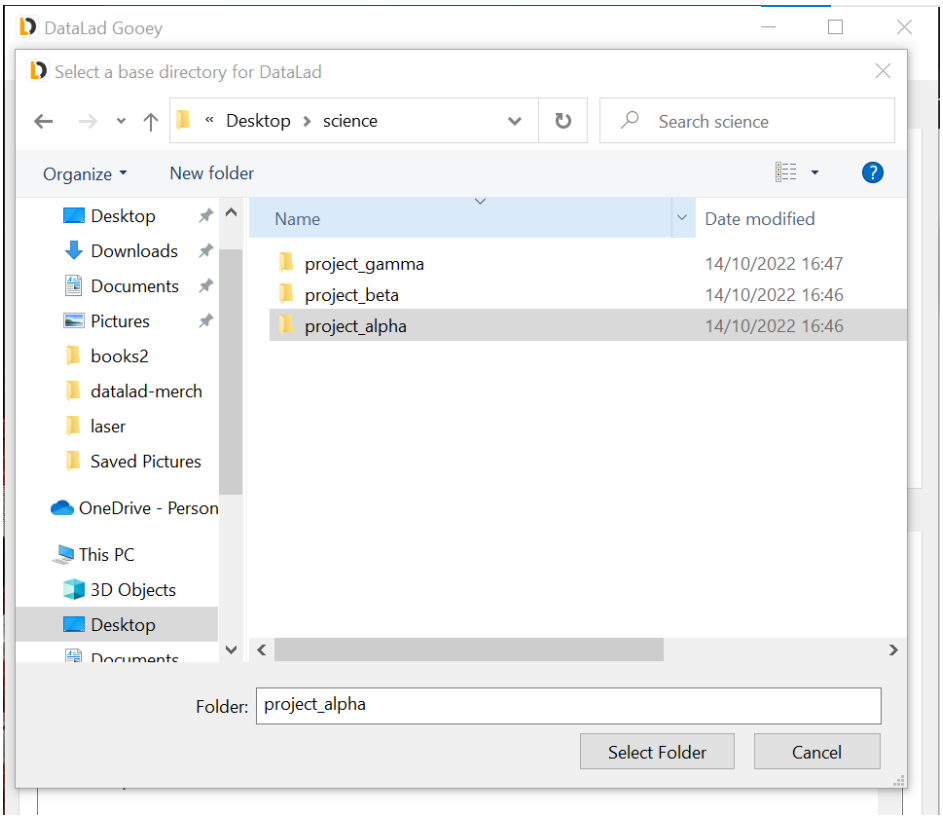
This section walks through the first steps to take when using `DataLad Gooey` and describes its main pieces and functions.

In order to start up `DataLad Gooey`, double-click the program's icon on your Desktop or in your Explorer if you are on Windows, find it among your Desktop files if you are using Linux, or launch it from the Terminal if you are on Mac[1]. An initial dialog lets you select a base directory. Navigate to any directory you would like to open up, select it, and click `Select Folder`.



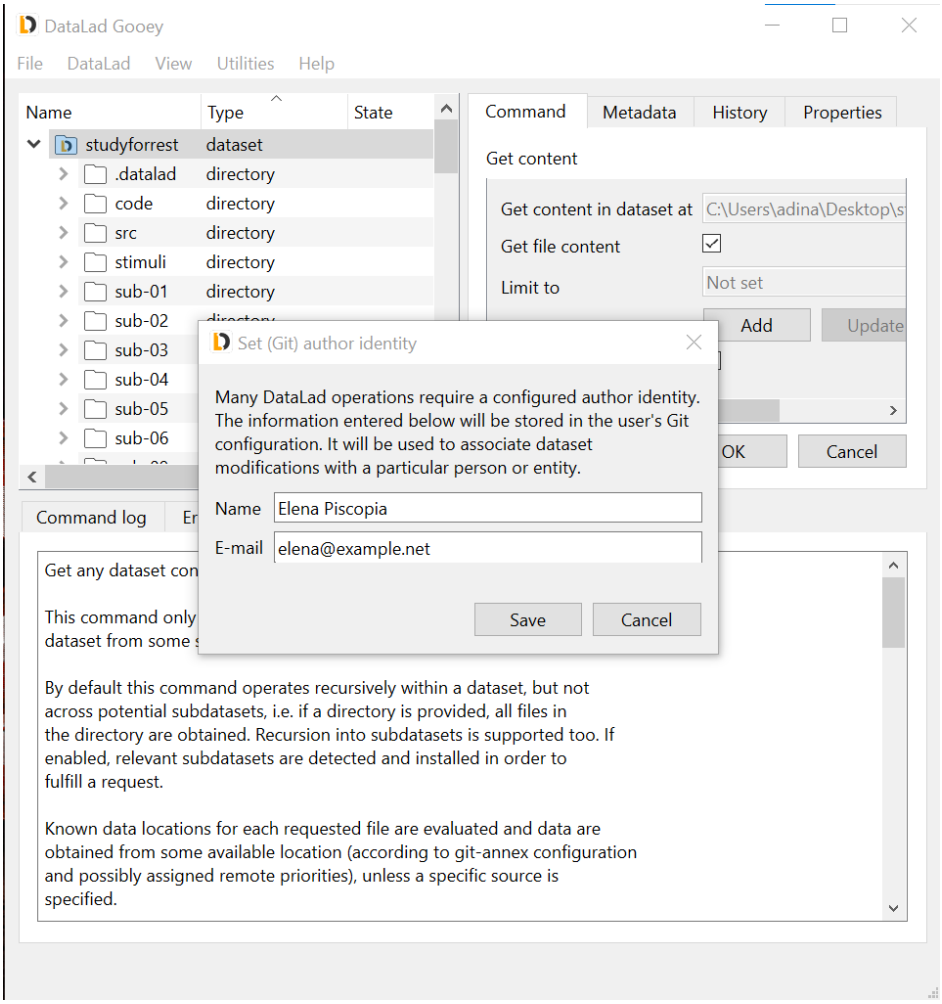

[1] Regardless of the operating system you are using, if you used `pip` to install `datalad gooey` you can also start it up from the command line, running `datalad gooey`. The optional `--path` argument lets you specify the root directory.

If you `Cancel` this dialog, the DataLad Gooey will open your home directory. The root directory can be changed at any later point using the `File` → `Set base directory` submenu from the top task bar, or from the right-click context menus of directories in the file tree.




### 2.2.1 Initial configuration

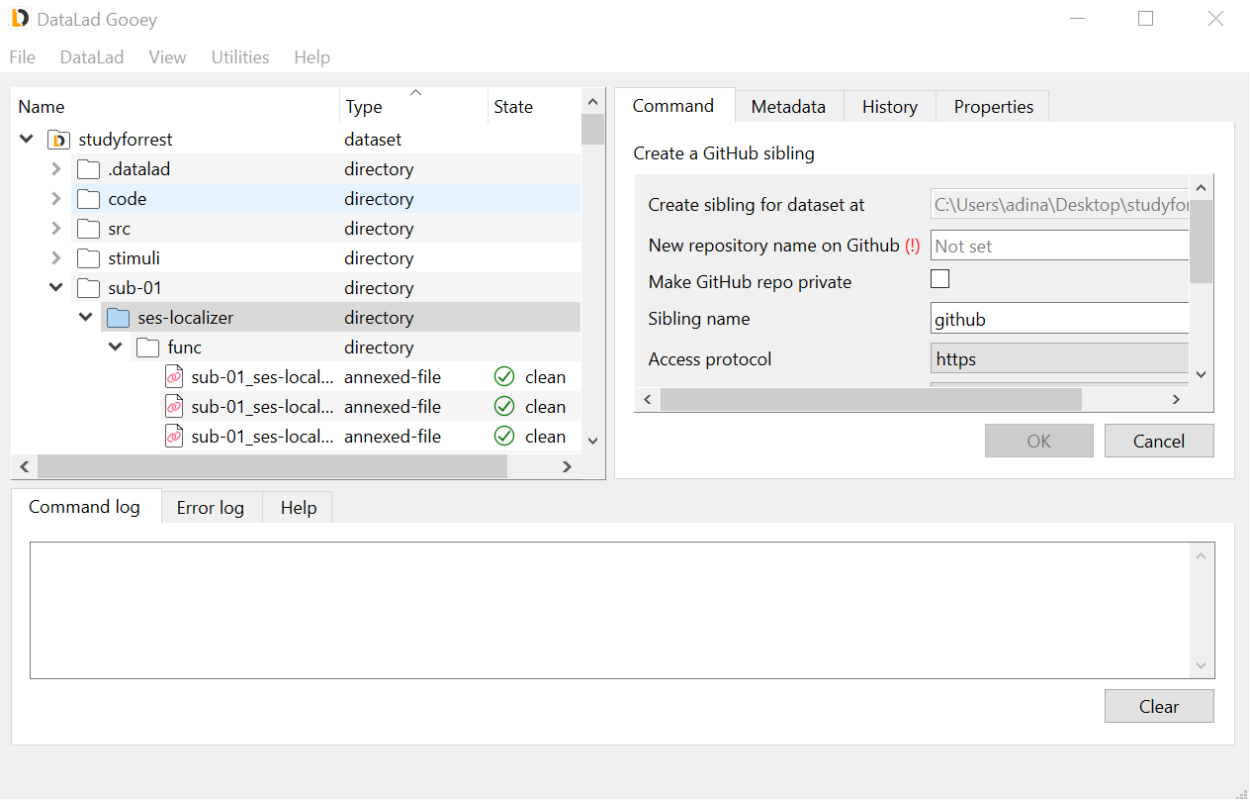
If you are using DataLad and Git for the first time on your computer, make sure that the first thing you do is to set your Git identity by clicking `Utilities` → `Set author identity` in the tab at the top of the application. Add your name and email address in the resulting dialog. This information will be used to include author information to the commands you will be running, and is required for many commands.




## 2.2.2 Application Overview

### The User Interface

In general, the DataLad Gooey interface has three main sections: A tree view on the upper left, pane on the upper right containing `Command`, `Metadata`, `History``and ``Properties` tabs, and the different log views at the bottom.

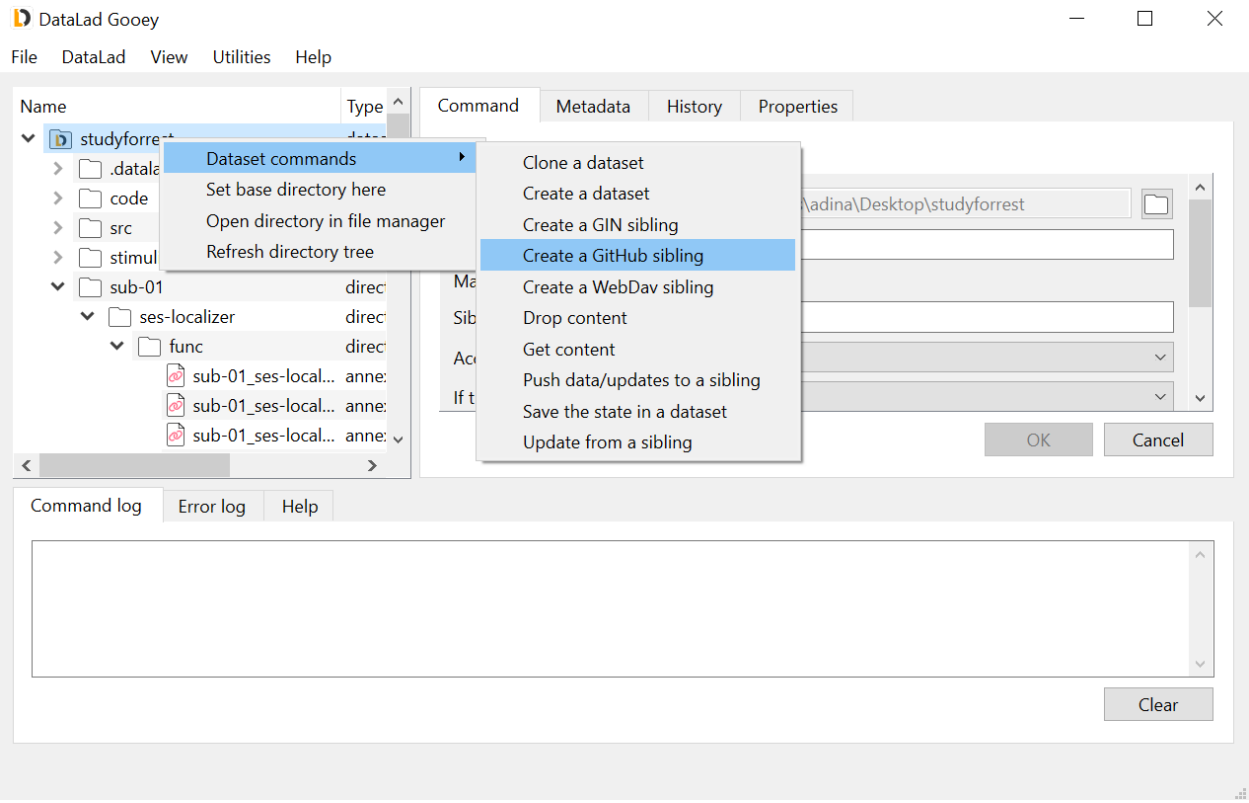



The tree view should display files and directories on your computer, starting from the root directory picked at start up. Double-clicking directories allows you to expand their contents, and to navigate into directory hierarchies. Double-clicking files will attempt to open them in your systems configured default application. You will notice that the `Type` and `State` annotations next to the file and directory names reveal details about files and directories: You can distinguish directories and DataLad datasets and files. Within datasets, files are either `annexed-file`'s or `file`'s, depending on how these files are tracked in the dataset. The `State` property indicates the version-state of files, datasets, or directories: A new file, for example, would be annotated with an `untracked` state, a directory with a newly added unsaved change would be `modified`, and neatly saved content would be annotated with a `clean` tag.

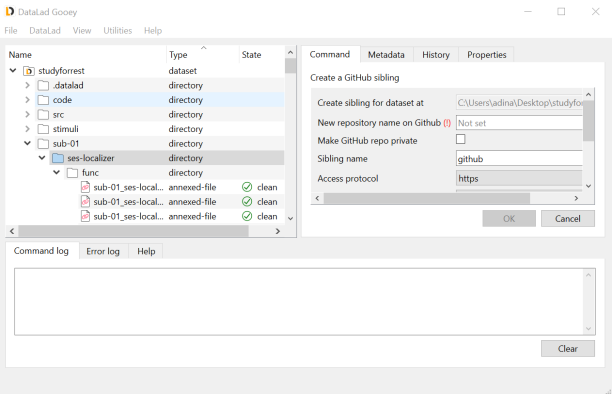
In addition to the information in the tree view, the `Properties` and `History` tab will load information for any selected directory or file. The `History` tab displays past commits associated with the file or directory, and the `Properties` tab displays known information such as the annex key or file size of annexed files.

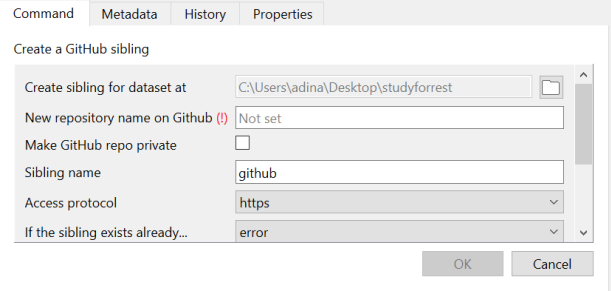
### Running a DataLad command

There are two ways of running DataLad command: either through the `Dataset` menu at the top, or by right-clicking on files, directories, or datasets in the tree view. The latter option might be simpler to use, as it only allows commands suitable for the item that was right-clicked on, and prefills many parameter specifications. The screenshot below shows the right-click context menu of a dataset, which has more available commands than directories or files.

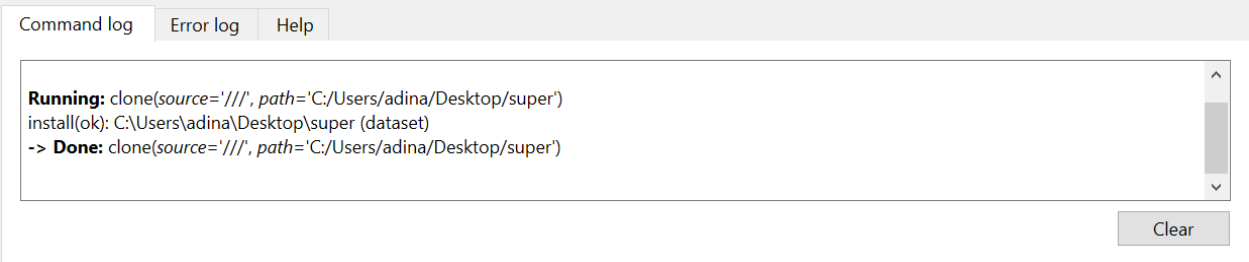



Once a DataLad command is selected, the `Command` tab contains relevant parameters and configurations for it. The parameters will differ for each command, but hovering over their submenus or editors will show useful hints what to set them to, and the `Help` tab below displays the commands detailed documentation. Little system-specific icons in the command panel can help you identify required input, or parameters that were wrongly specified (see detailed screenshot on the right).





Once input validation passes on all parameters, the `OK` button will become functional and will execute the command. The `Command log` will continuously update you on the state of running and finished commands, displaying, where available, progress bars, result reports, or command summaries.




During command execution, a small hammer symbol lets you interrupt ongoing commands. If you accidentally started to `get` hundreds of Gigabytes worth of files instead of only one directory, clicking this button will stop the command.


---

**Note:** The stop button will stop the execution at the next possible chance, which is after the next part of the command (such as the retrieval of the next file) has finished. This means it is not an immediate stop like a `CTRL-C` press in the command line, but a safe interruption. This also means that stopping might take longer than you expect, depending on the command that gets interrupted.

---

Should a command fail, a detailed traceback with details about the failure will be send to the `Error log` tab right next to the `Command log`. You can use the information from this tab to investigate and fix problems.

### Navigation

The interface can be navigated via mouse clicks, or, on most operating systems, via keyboard shortcuts as well. Low lines under specific letters of menus or submenus identify the shortcut[2]. Accessing the shortcut to a menu requires pressing `Alt` and the respective letter: `Alt + f` for example will open the `File` menu. Pressing further letters shortcuts to submenu actions: `Alt + f + q` will shortcut to `Quit` and close the application, while `Alt + d + g` will open a `get` command in the Command panel.

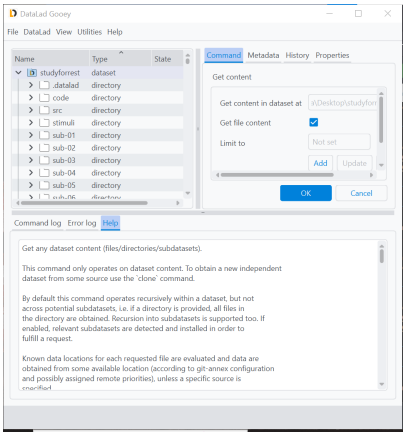
In addition, path parameters (such as the `dataset` parameter) can be filled via drag and drop from your system's native file browser.

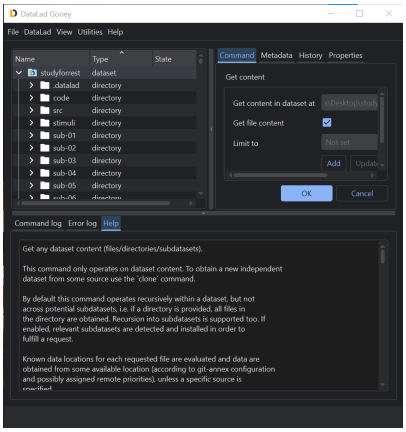
### The View Menu

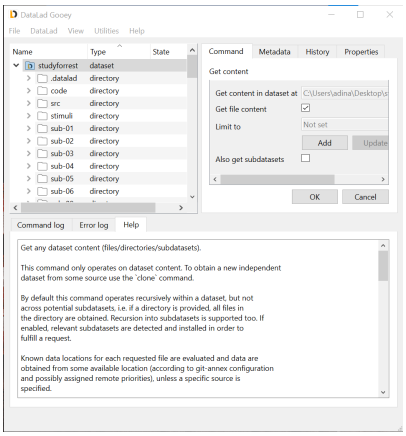
The `View` menu contains two submenus that allow you to alter the appearance of the interface. Whenever you change the appearance of the interface, you need to close and reopen the program in order to let the change take effect.

The `Theme` submenu lets you switch between a light, dark, and system theme, shown below in order.

[2] Windows users may not automatically see underlined letters. To make them visible, press the `Alt` key. Mac users won't see underlined letters as it would violate the guidelines of macOS graphical user interface aqua.







The `Suite` submenu lets you switch between suites that alter the command selection. The two suites you will always be able to select between is a "simplified" command set, reduced to the most essential commands and parameters, and a "complete" command set, which is a development preview. DataLad extensions can add additional suites when you install them. Please note that we recommend the "simplified" command suite to users, as the complete suite can contain experimental implementations.

### The Utilities and Help Menu

The `Utilities` menu has a several useful functions. One is a convenience version checker that can tell you whether there is a newer DataLad version available.

---

**Note:** The `Check for new version` feature requires a network connection.

---

The `Manage credentials` submenu opens an interface to the Credential manager. The section *Credential management with DataLad Gooey* has a detailed overview. The `Set author identity` submenu lets you set and change your Git identity and was already covered above.

The `Help` menu contains a range of actions to find additional information or help. `Report a problem` contains links for filing issues and getting in touch with the developers. `Diagnostic infos` will create a report about the details of your installation and system that you can copy-paste into such issues.

## 2.3 Central Concepts (Glossary)

Working with DataLad is easier if you know a few technical terms that are regularly referred to in the documentation and the GUI interface. This glossary provides short definitions, and links relevant additional documentation, where available.

**annex**
: *git-annex* concept: a different word for the internal location in a dataset that *annexed-file*'s are version controlled in.

**annexed-file**
: Files managed by *git-annex* are annotated as "annexed-file". Annexed files have access to additional commands in their context menus such as *get* and *drop*.

**branch**
: Git concept: A lightweight, independent history streak of your dataset. Branches can contain less, more, or changed files compared to other branches, and one can merge the changes a branch contains into another branch. DataLad Gooey only views the currently checked out branch in your dataset, and does not support Git commands that expose branching functionality.

**clone**
: The datalad clone command retrieves a copy of a *Git* repository or *DataLad dataset* from a local or remote path or URL. In Git-terminology, all "installed" datasets are clones.

**commit**
: Git concept: Adding selected changes of a file or dataset to the repository, and thus making these changes part of the revision history of the repository. The datalad save command creates a commit in the selected dataset. Commits should always have an informative *commit message*.

**commit message**
: Git concept: A concise summary of changes you should attach to a **`datalad save`** command. This summary will show up in your *DataLad dataset* history.

**DataLad dataset**
: 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.

**DataLad extension**
: Python packages that equip DataLad with specialized commands. The section *extensions_intro <http://handbook.datalad.org/en/latest/r.html?extensions>_* of the DataLad Handbook gives and overview of available extensions and links to Handbook chapters that contain demonstrations.

**DataLad subdataset**
A DataLad dataset contained within a different DataLad dataset (the parent or *DataLad superdataset*).

**DataLad superdataset**
A DataLad dataset that contains one or more levels of other DataLad datasets (*DataLad subdataset*).

**drop**
The datalad drop command drops file content of annexed files. It is the antagonist to *get*.

**GIN**
A web-based repository store for data management that you can use to host and share datasets. Find out more about GIN here.

**Git**
A version control system to track changes made to small-sized files over time. You can find out more about git in this (free) book or these interactive Git tutorials on *GitHub*.

**git-annex**
A distributed file synchronization system, enabling sharing and synchronizing collections of large files. It allows managing files with *Git*, without checking the file content into Git.

**git-annex branch**
A *branch* that exists in your dataset, if the dataset contains an annex. The git-annex branch is completely unconnected to any other branch in your dataset, and contains different types of log files. Its contents are used for git-annex's internal tracking of the dataset and its annexed contents. DataLad Gooey provides support for adding git annex metadata, but does not otherwise support operations on dataset branches

**GitHub**
GitHub is an online platform where one can store and share version controlled projects using Git (and thus also DataLad project). See`GitHub.com <https://github.com/>`_.

**GitLab**
An online platform to host and share software projects version controlled with *Git*, similar to *GitHub*. See Gitlab.com.

**get**
The datalad get command gets file content of annexed files. It is the antagonist to *drop*.

**https**
Hypertext Transfer Protocol Secure; A protocol for file transfer over a network.

**pip**
A Python package manager. Short for "Pip installs Python". `pip install <package name>` searches the Python package index PyPi for a package and installs it while resolving any potential dependencies.

**remote**
Git-terminology: A repository (and thus also *DataLad dataset*) that a given repository tracks. A *sibling* is DataLad's equivalent to a remote.

**SSH**
Secure shell (SSH) is a network protocol to link one machine (computer), the *client*, to a different local or remote machine, the *server*.

**SSH key**
An SSH key is an access credential in the SSH protocol that can be used to login from one system to remote servers and services, such as from your private computer to an SSH server, without supplying your username or password at each visit. To use an SSH key for authentication, you need to generate a key pair on the system you would like to use to access a remote system or service (most likely, your computer). The pair consists of a *private* and a *public* key. The public key is shared with the remote server, and the private key is used to authenticate your machine whenever you want to access the remote server or service. Services such as *GitHub*, *GitLab*, and

*GIN* use SSH keys and the SSH protocol to ease access to repositories. This tutorial by GitHub is a detailed step-by-step instruction to generate and use SSH keys for authentication.

**sibling**
DataLad concept: A dataset clone that a given *DataLad dataset* knows about. Changes can be retrieved and pushed between a dataset and its sibling. It is the equivalent of a *remote* in Git.

**version control**
Processes and tools to keep track of changes to documents or other collections of information.

## 2.4 Walk-through: Dataset hosting on GIN

In this walkthrough, we will use DataLad Gooey to create a dataset, save it contents, and publish it to GIN (G-Node Infrastructure).

### 2.4.1 Prerequisites

In order to use GIN for hosting and sharing your datasets, you need to:

- Register a GIN account;
- add a personal access token to your GIN account (for creation of repositories with DataLad);
- add an SSH key to your GIN account (for uploading annexed contents).

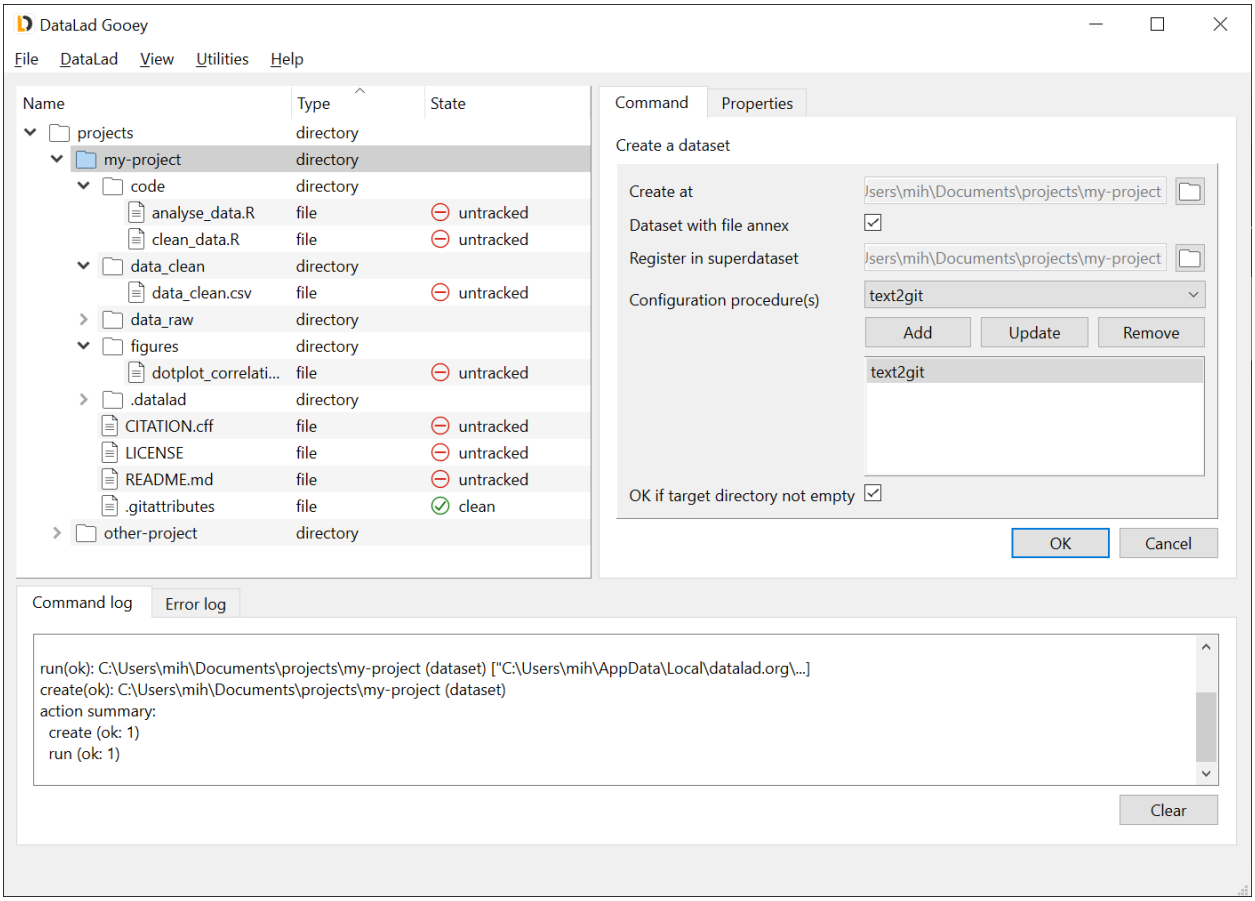
Follow the instructions on GIN to do so.

If you need to generate an SSH key pair and want to stay in the world of graphical interfaces, we recommend using PuTTYgen for this purpose. Your private key needs to be placed in the *.ssh* folder in your home directory for it to be picked up correctly.

### 2.4.2 Create a dataset

Let's assume that we are starting with an existing folder which already has some content, but is not yet a DataLad dataset. Let's open the DataLad Gooey and set a base directory to our folder, or its parent directory.

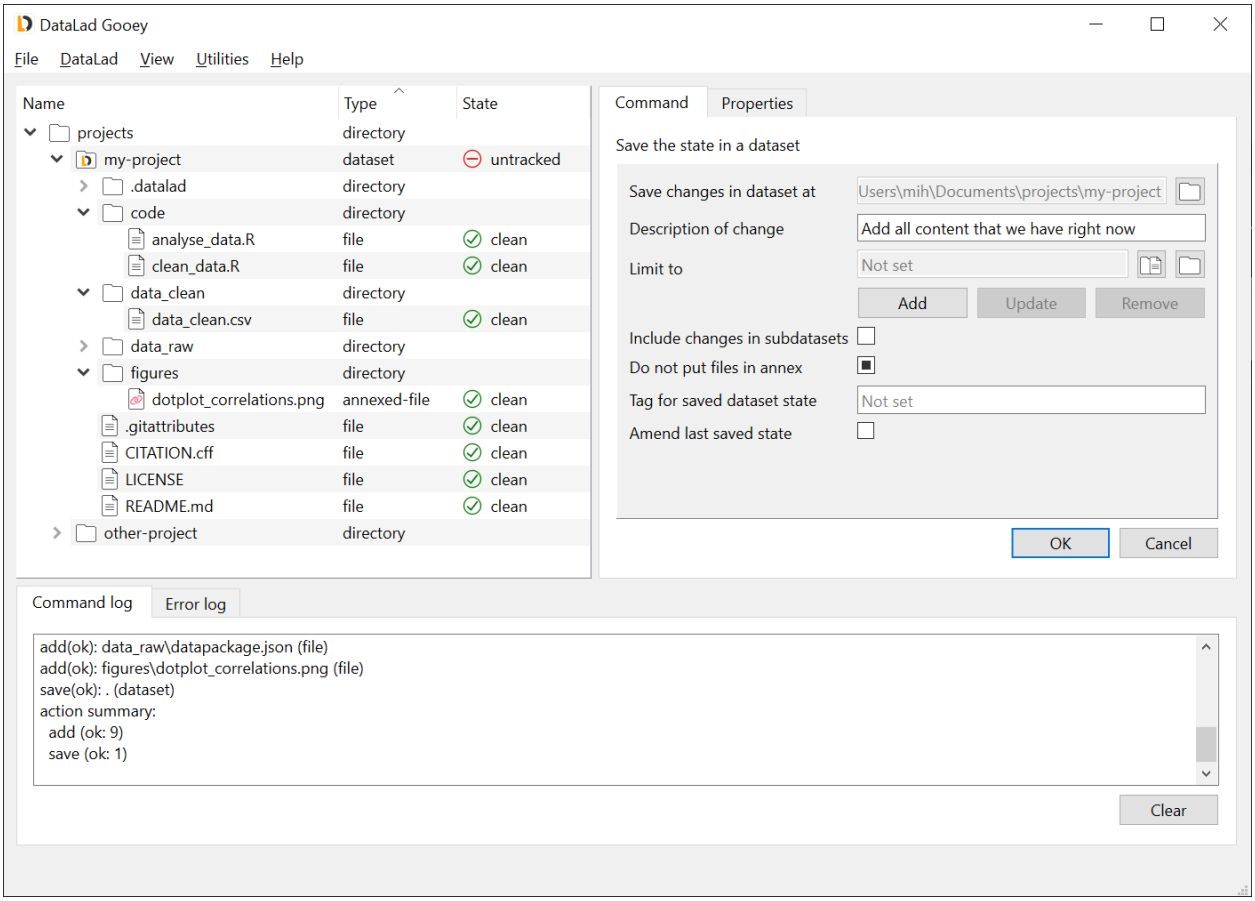
Our first operation is to to create a DataLad dataset. For this, right-click your folder and select *Directory commands* → *Create a dataset*. This will populate the Command tab on the right with options for the selected command. The first value (*Create at*) is already populated, since we used right-click to issue the command. We leave *Dataset with file annex* checked (default), and *Register in superdataset* not set (default). In this example we want to configure our dataset to annex binary, but not text files. To do so, select *text2git* from the list of *Configuration procedure(s)* and click *Add*. Finally, check the *OK if target directory not empty* to enforce dataset creation out of a non-empty folder. With the options selected, click *OK*.




### 2.4.3 Save the contents

Right-click the newly created dataset, and select *Dataset commands → Save the state in a dataset*. Parameters required for the Save command should appear in the Command tab. Fill in the *Description of change* (this is the commit message associated with the save). Leave all other fields default (note: *Do not put files in annex* is greyed out, not checked, i.e. it has no value). Here, we are saving all files at once, but if we wanted we could limit the save operation to selected files, or trigger it by clicking on a specific file. Once ready, click *OK*.

Note that after this operation, "untracked" files changed their state to "clean". Different from files, the dataset state is still "untracked", because it is not registered in any superdataset. Because we used the *text2git* configuration, only the PNG file changed its type to "annexed-file" in the screenshot below.



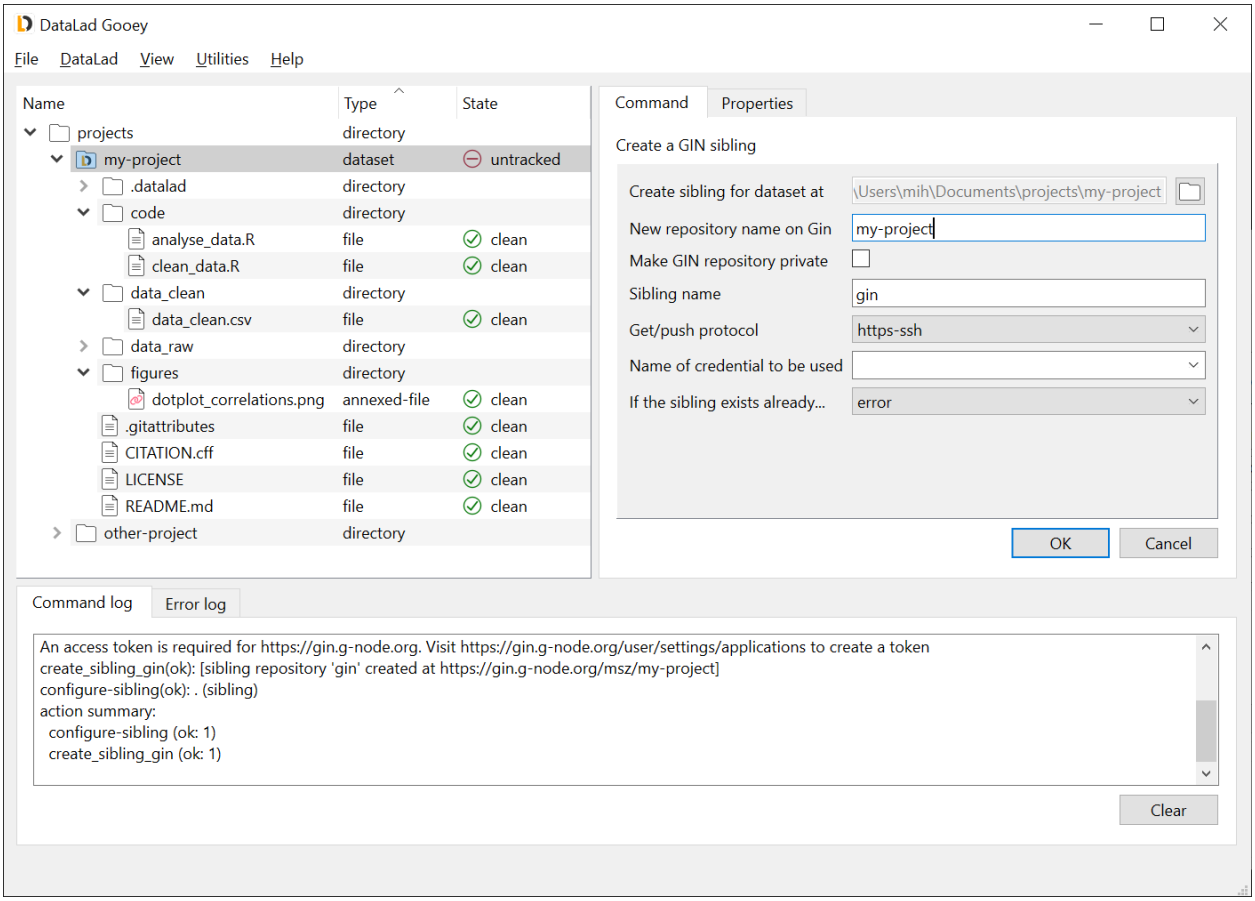

### 2.4.4 Create a GIN sibling

Creating a GIN sibling will create a new repository on GIN, and configure your dataset with its address. To perform this action, right-click your dataset, and select *Dataset commands* → *Create a GIN sibling*. Fill in the *New repository name on Gin* (and, optionally, check the *Make GIN repository private*). You can leave all other options default.

In the *Name of the credential to be used* field, you can pick previously used credentials. If no value is given, and no previous credentials exist, the credentials will be save with website name (*gin.g-node.org*) by default.

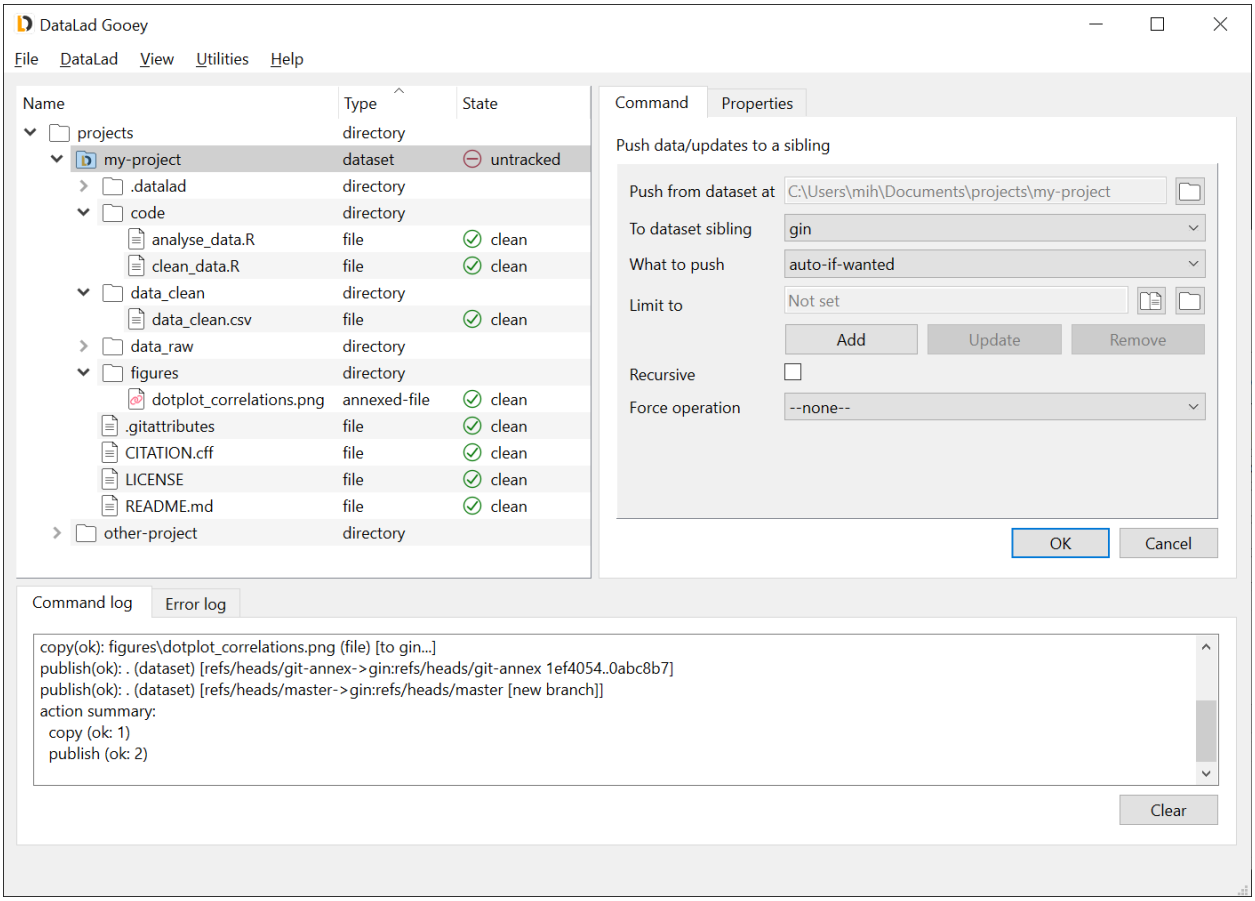
Click *OK*.

At this point, a pop-up window will appear and you will be asked for a token. Paste the access token generated from GIN website, and click *OK*.




### 2.4.5 Push to the GIN sibling

Right-click *Dataset commands* → *Push data/updates to a sibling*. The only thing you need to select is the value of *To dataset sibling* - this will be the sibling name from the step above. Leave other options default, and click *OK*.




### 2.4.6 Retrieve the data from GIN

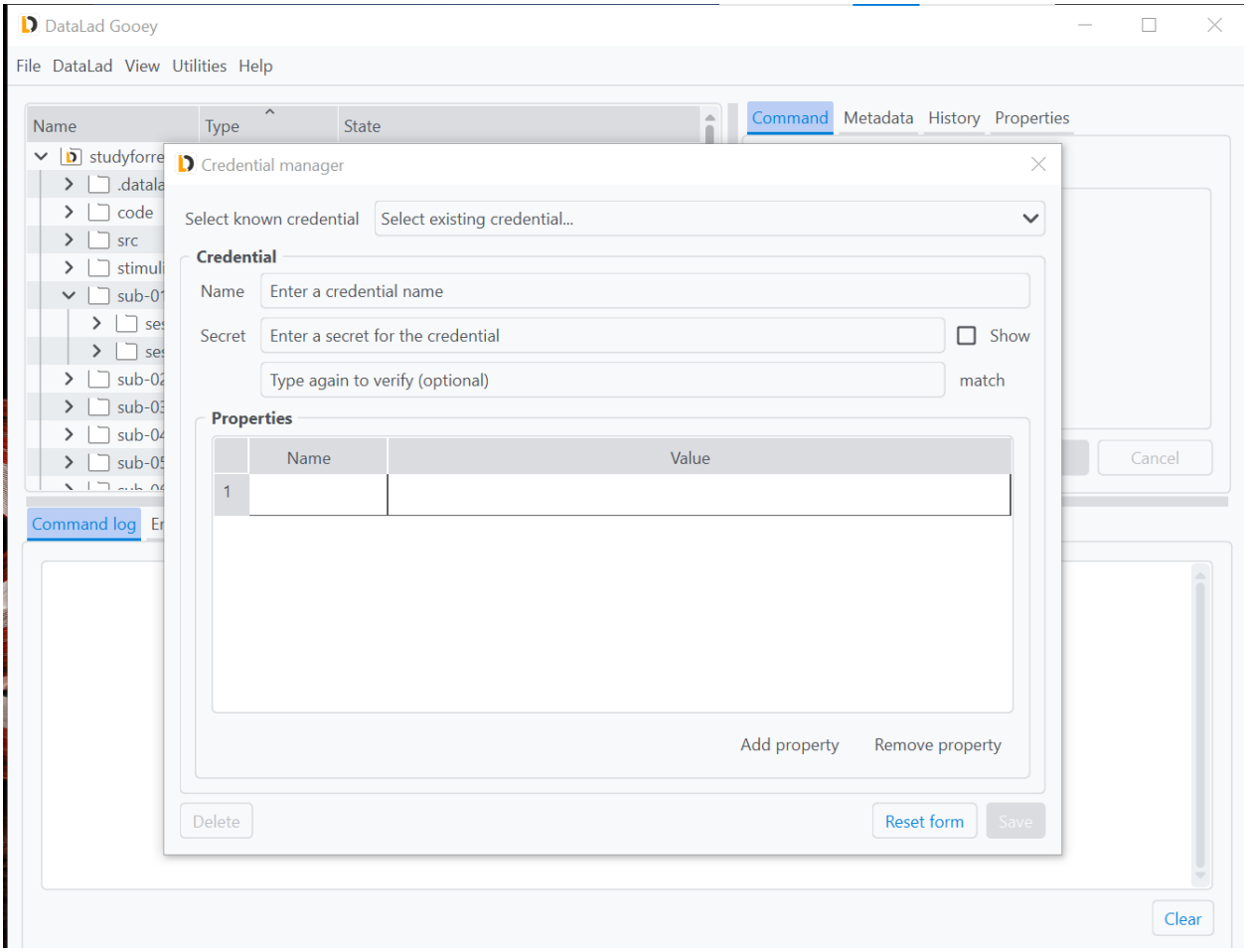
Finally we can confirm that our dataset can be obtained from GIN (possibly by other users who have access). Let's try making a clone in the same parent directory as our dataset, but under a different name. To do this, from the menu select *DataLad → Clone a dataset*. In the *Clone from* field, enter the dataset URL taken from GIN. Here, you can use either the HTTPS (for public repositories) or SSH (for private repositories) URL. Note that when using HTTPS, you need to remove the `.git` from the URL ending for proper interaction with GIN. Next, click the directory selection icon to the right of *Clone into* field, and use the directory picker to create and select a new directory named, for example, `cloned-dataset`. Afterwards, click *OK*.

To obtain the annexed contents in the cloned dataset, right click it in the file tree, select *Dataset commands → Get content*, and click *OK*. With other options kept default, this will download all annexed content in the dataset; if you wanted to obtain contents selectively, you could use the *Limit to* option. Alternatively, you could right-click individual files, and use *File commands → Get content*.

## 2.5 Credential management with DataLad Gooey

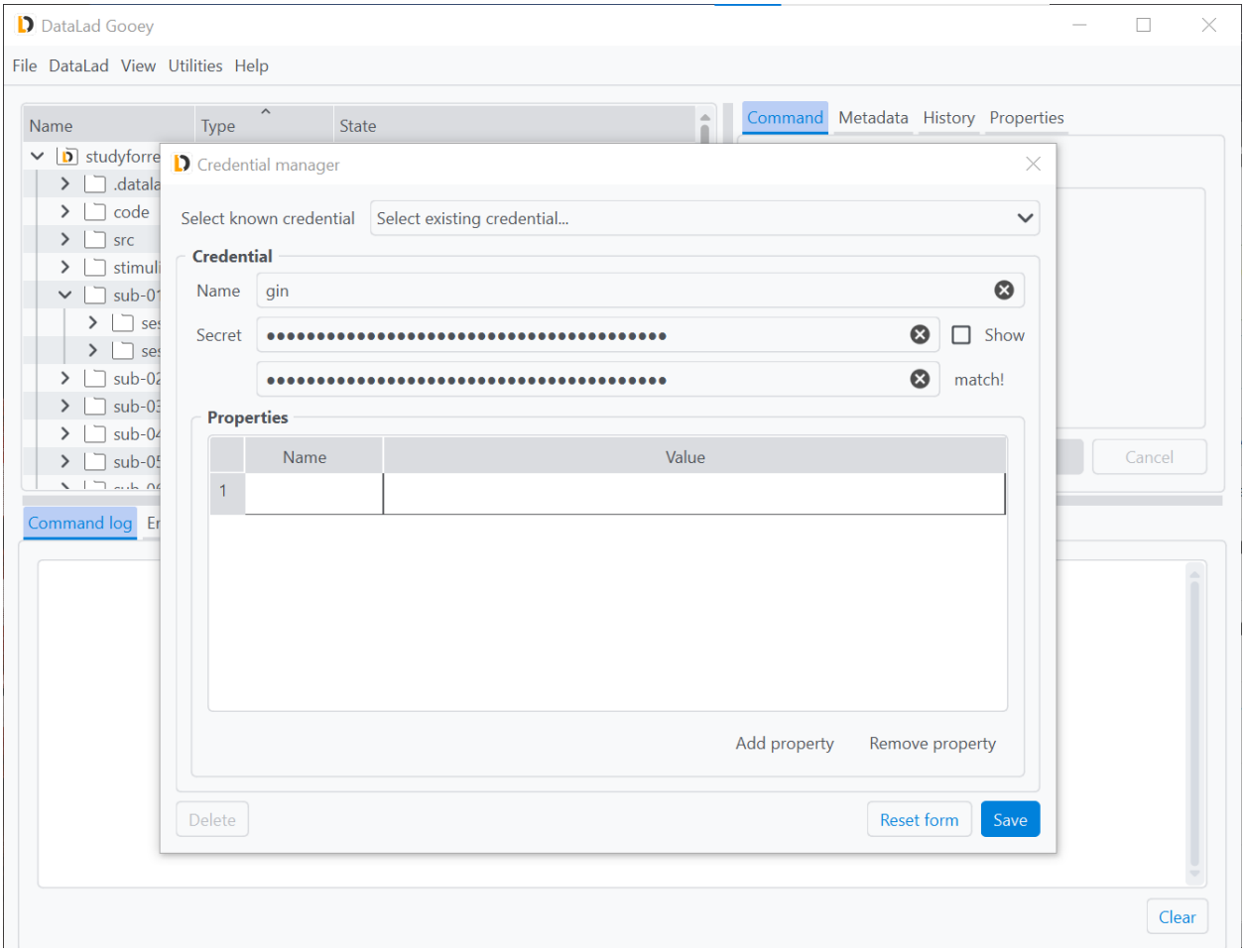
DataLad can store credentials and use them to authenticate against a variety of services or infrastructures. Among them are, for example, the credentials required to push datasets to siblings on Gin, GitHub, or similar services. `DataLad Gooey`'s credential manager makes setting, querying, and removing credentials much easier than performing the actions via the command line. It is a front-end for the modernized datalad credential command from datalad-next.

Access to this tooling is provided via the `Utilities` -> `Manage credential` menu in the tab bar.

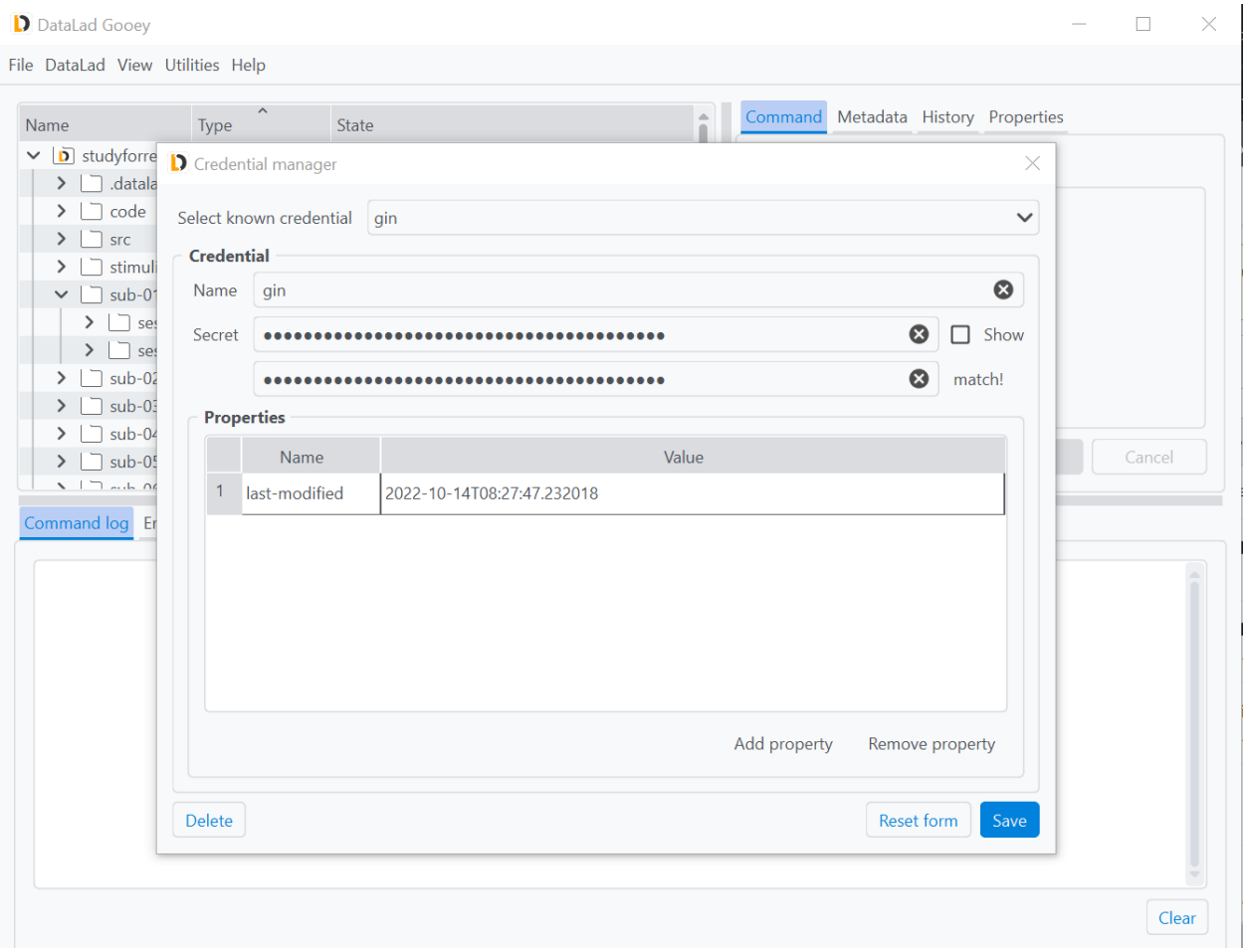



All credentials are identified via a unique name, and should at least contain a secret such as a token, a password, or a key. The `Select known credential` drop-down menu will list all known secrets for query or modification. This drop-down menu can be helpful to discover credentials, update them after their secrets changed, or remove credentials that became obsolete via the `Delete` button at the bottom of the form.

In order to create a new credential, for example a token to authenticate against *Gin*, enter a name of your choice (here it is `gin`) and a corresponding secret in the `Credential` tab underneath. As the `Secret` field is confidential, the characters you type or paste will be hidden, but the `show` button on the right can be ticked to view it in clear text. To prevent typos, the secret has to be repeated and match the previous entry. The `Save` button at the bottom will store the secret in your systems credential store.

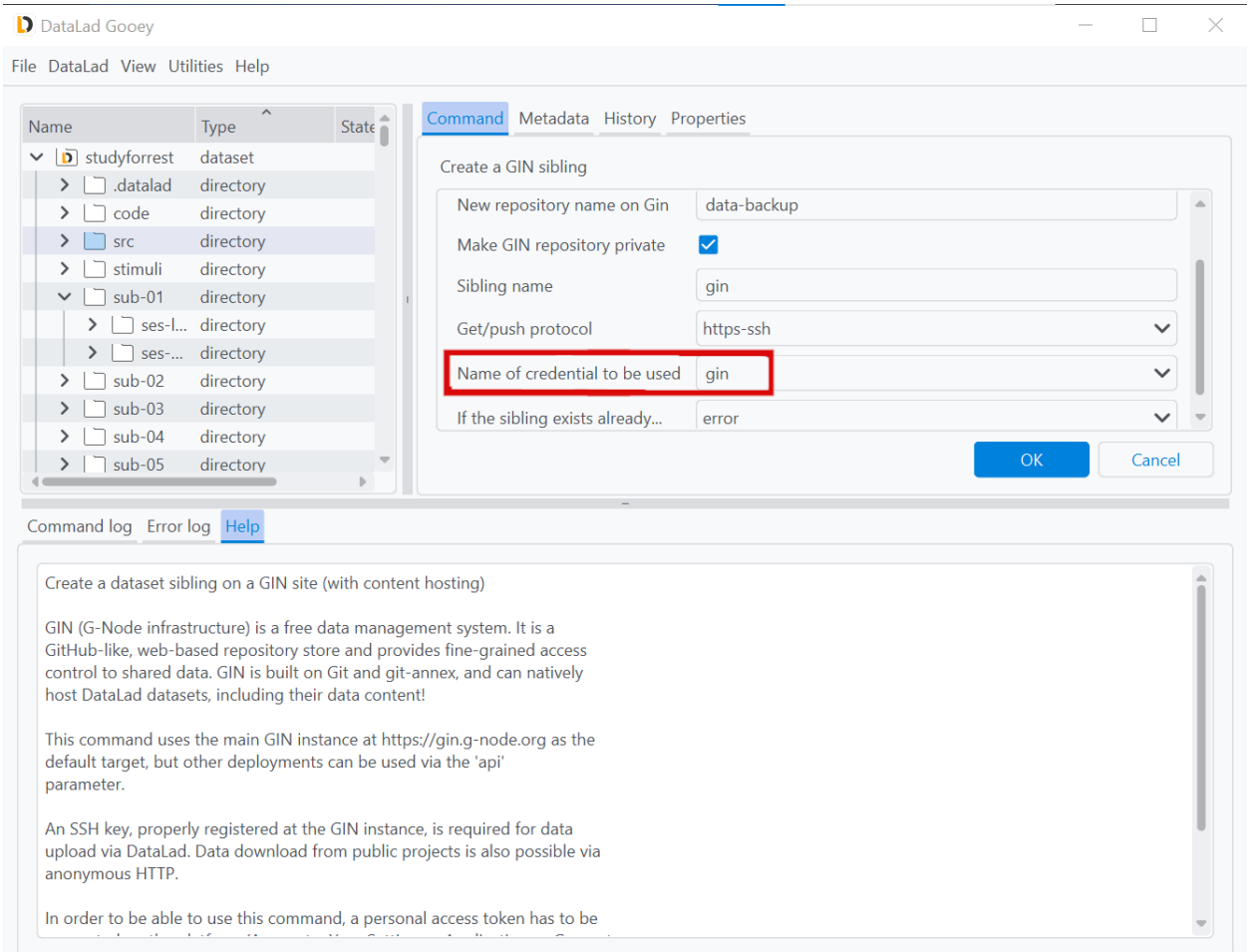



Credentials can have properties denoting additional details about them useful for queries or required for authentication to particular services. Once saved, the `gin` credential gained an automatic property, the `last-modified` property. You can add additional arbitrary properties or alter or remove existing ones in the central `Properties` tab.



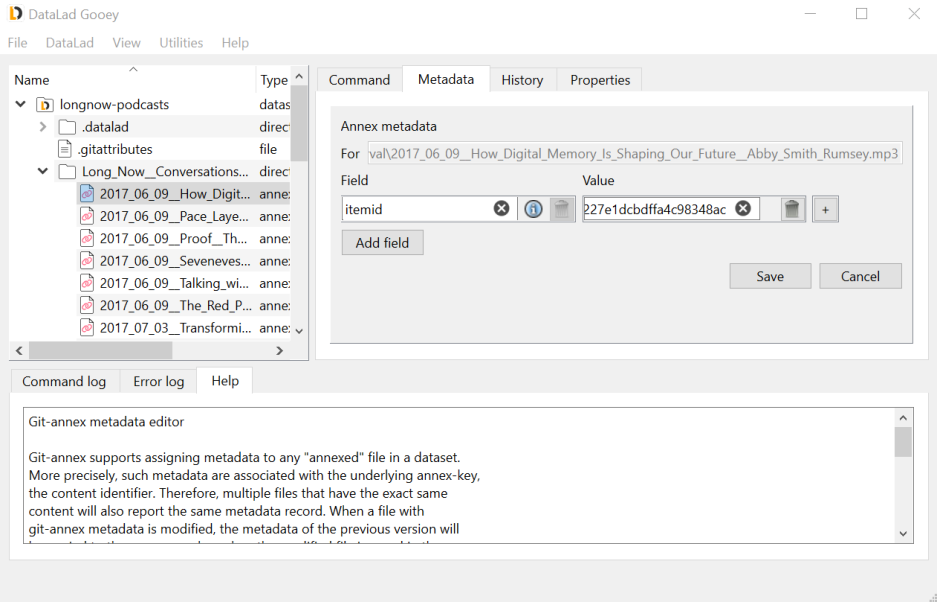

Once saved, the credential will be available in all commands with a credential parameter via a drop-down menu for you to select. This ensures that you can precisely select which credential is used in every operation. Especially if you have several accounts on one and the same service or several authentication methods with a different set of permissions, such as one for your private account and one for your organizations account, this comes in handy.

In the screenshot below, the newly created credential `gin` is used in the parametrization of a `create-sibling-gin` command.




## 2.6 Setting git-annex metadata

*git-annex* has its own concept and implementation of file metadata, and DataLad Gooey provides a graphical user interface for querying, removing, and setting metadata. This functionality is exposed via the `Metadata` tab or right-click context menue on any annexed file.

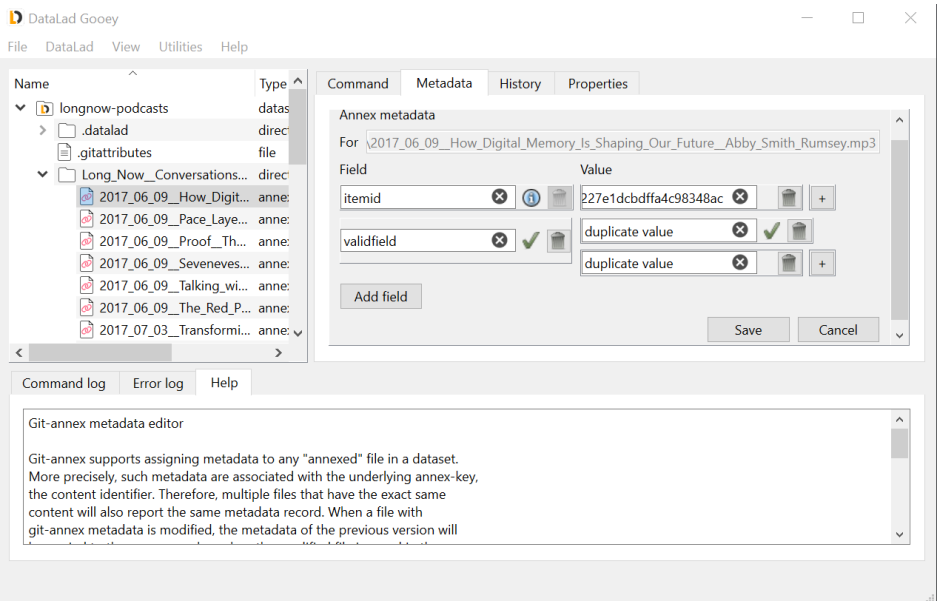



Each annexed file's content[1] can have any number of metadata *fields* attached to it to describe it, and each field can have any number of *values*. This metadata is stored internally, in the *git-annex branch*.

The Annex metadata editor will show all existing fields and values for a selected file, and let users add additional fields and values as well as deleting them. A subsequent `Save` will write the provided annex metadata to the file.

### 2.6.1 Input validation

git-annex's metadata is highly flexible and allows arbitrary content, but it is impossible to have two fields of the same name, or two identical values belonging to one field. This is validated automatically after each entry, and only valid metadata can be saved. Icons attached to conflicting fields will let you know which metadata elements need fixing.




[1] Note that metadata is attached to file content, not file names, i.e. the git-annex key corresponding to the content of a file, not to a particular filename on a particular git branch. This means that all files with the same key share the same metadata, which is stored in the *git-annex branch*. If a file is modified, the metadata of the previous version will be copied to the new key when git-annex adds the modified file.

## 2.7 Setting Metadata

Coming soon - stay tuned!

CHAPTER

THREE

# COMMANDS AND API

## 3.1 Command line reference

This module reference extends the manual with an overview of the available functionality built into datalad gooey. Please note that apart from the main command, the commands this module provides are internal helpers used within DataLad Gooey for fast annotations and file tree overviews. Unlike the DataLad core package or its extensions, the commands provided by DataLad Gooey are thus not intended to be used directly.

### 3.1.1 datalad gooey

#### Synopsis

```
datalad gooey [-h] [-p PATH] [--postinstall] [--version]
```

#### Description

DataLad GUI

Long description of arbitrary volume.

*Examples*

Launch the DataLad Graphical User Interface (GUI, a.k.a Gooey) at the specified location.:

```
% datalad gooey --path 'path/to/root/explorer/directory'
```

#### Options

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

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

#### -p *PATH*, --path *PATH*

The root location from which the Gooey file explorer will be launched (default is current working directory).

#### --postinstall

Perform post-installation tasks.

#### --version

show the module and its version which provides the command

#### Authors

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

### 3.1.2 datalad gooey-askpass

#### Synopsis

```
datalad gooey-askpass [-h] [--version]
```

#### Description

Internal helper for datalad-gooey

#### 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>.

### 3.1.3 datalad gooey-lsdir

#### Synopsis

```
datalad gooey-lsdir [-h] [--version] path
```

#### Description

Internal helper for datalad-gooey

#### Options

#### path

#### -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>.

### 3.1.4 datalad gooey-status-light

#### Synopsis

```
datalad gooey-status-light [-h] [-d DATASET] [--version] path
```

#### Description

Internal helper for datalad-gooey

#### Options

#### path

#### -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

#### --version

show the module and its version which provides the command

#### Authors

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

## 3.2 Python module reference

This module reference extends the manual with an overview of the available functionality built into datalad gooey. Please note that apart from the main command, the commands this module provides are internal helpers used within DataLad Gooey for fast annotations and file tree overviews. Unlike the DataLad core package or its extensions, the commands provided by DataLad Gooey are thus not intended to be used directly.

| | |
|---|---|
| *gooey*([path, postinstall]) | DataLad GUI |
| *gooey_askpass*() | Internal helper for datalad-gooey |
| *gooey_lsdir*(path) | Internal helper for datalad-gooey |
| *gooey_status_light*([dataset, path]) | Internal helper for datalad-gooey |

### 3.2.1 datalad.api.gooey

datalad.api.**gooey**(*path: str = None*, *postinstall: bool = False*)

DataLad GUI

Long description of arbitrary volume.

#### Examples

Launch the DataLad Graphical User Interface (GUI, a.k.a Gooey) at the specified location.:

```
> gooey(path='path/to/root/explorer/directory')
```

**Parameters**

- **path** -- The root location from which the Gooey file explorer will be launched (default is current working directory). [Default: None]
- **postinstall** (*bool, optional*) -- Perform post-installation tasks. [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']

### 3.2.2 datalad.api.gooey_askpass

datalad.api.**gooey_askpass**()

Internal helper for datalad-gooey

### 3.2.3 datalad.api.gooey_lsdir

datalad.api.**gooey_lsdir**(*path: Path*)

Internal helper for datalad-gooey

**Parameters**

- **path** --
- **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']

### 3.2.4 datalad.api.gooey_status_light

datalad.api.**gooey_status_light**(*dataset=None*, *path: Path = None*)

Internal helper for datalad-gooey

**Parameters**

- **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]
- **path** -- [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']

# INDEX