# renku Documentation

***Release 0.6.1***

**Swiss Data Science Center**

**Jan 21, 2020**

# Contents

A Python library for the Renku collaborative data science platform. It allows the user to create projects, manage datasets, and capture data provenance while performing analysis tasks.

**NOTE:** `renku-python` is the python library for Renku that provides an SDK and a command-line interface (CLI). It *does not* start the Renku platform itself - for that, refer to the Renku docs on running the platform.

CHAPTER 1

# Installation

The latest release is available on PyPI and can be installed using `pip`:

```
$ pip install renku
```

The latest development versions are available on PyPI or from the Git repository:

```
$ pip install --pre renku
# - OR -
$ pip install -e git+https://github.com/SwissDataScienceCenter/renku-python.git
↪#egg=renku
```

Use following installation steps based on your operating system and preferences if you would like to work with the command line interface and you do not need the Python library to be importable.

## 1.1 Homebrew

The recommended way of installing Renku on MacOS and Linux is via Homebrew.

```
$ brew tap swissdatasciencecenter/renku
$ brew install renku
```

## 1.2 Isolated environments using `pipx`

Install and execute Renku in an isolated environment using `pipx`. It will guarantee that there are no version conflicts with dependencies you are using for your work and research.

Install pipx and make sure that the `$PATH` is correctly configured.

```
$ python3 -m pip install --user pipx
$ pipx ensurepath
```

Once `pipx` is installed use following command to install `renku`.

```
$ pipx install renku
$ which renku
~/.local/bin/renku
```

Prevously we have recommended to use `pipsi`. You can still use it or migrate to **pipx**.

## 1.3 Docker

The containerized version of the CLI can be launched using Docker command.

```
$ docker run -it -v "$PWD":"$PWD" -w="$PWD" renku/renku-python renku
```

It makes sure your current directory is mounted to the same place in the container.

# CHAPTER 2

# Getting Started

Interaction with the platform can take place via the command-line interface (CLI).

Start by creating for folder where you want to keep your Renku project:

```
$ mkdir -p ~/temp/my-renku-project
$ cd ~/temp/my-renku-project
$ renku init
```

Create a dataset and add data to it:

```
$ renku dataset create my-dataset
$ renku dataset add my-dataset https://raw.githubusercontent.com/
↪SwissDataScienceCenter/renku-python/master/README.rst
```

Run an analysis:

```
$ renku run wc < data/my-dataset/README.rst > wc_readme
```

Trace the data provenance:

```
$ renku log wc_readme
```

These are the basics, but there is much more that Renku allows you to do with your data analysis workflows.

For more information about using *renku*, refer to the *renku –help*.

# CHAPTER 3

# Project Information

## 3.1 License



## 3.2 Authors

Python SDK and CLI for the Renku platform.


- Swiss Data Science Center <contact@datascience.ch>


## 3.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

### 3.3.1 Types of Contributions

#### Report Bugs

Report bugs at https://github.com/SwissDataScienceCenter/renku-python/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

#### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

#### Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

#### Write Documentation

Renku could always use more documentation, whether as part of the official Renku docs, in docstrings, or even on the web in blog posts, articles, and such.

#### Submit Feedback

The best way to send feedback is to file an issue at https://github.com/SwissDataScienceCenter/renku-python/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

### 3.3.2 Get Started!

Ready to contribute? Here's how to set up *renku* for local development.

1. Fork the *SwissDataScienceCenter/renku-python* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/renku.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv renku
$ cd renku/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

   Before you submit a pull request, please reformat the code using yapf.

```
$ yapf -irp .
```

   You may want to set up yapf styling as a pre-commit hook to do this automatically:

```
$ curl https://raw.githubusercontent.com/google/yapf/master/plugins/pre-commit.sh␣
↪-o .git/hooks/pre-commit
$ chmod u+x .git/hooks/pre-commit
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

### 3.3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

0. **Make sure you agree with the license and follow the [legal matter](https://github.com/SwissDataScienceCenter/documentation/wiki/Legal-matter).**
1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 3.5, 3.6 and 3.7. Check https://travis-ci.org/SwissDataScienceCenter/renku-python/pull_requests and make sure that the tests pass for all supported Python versions.

## 3.4 Changes

### 3.4.1 0.6.0 (2019-09-18)

#### Bug Fixes

- adds _label and commit data to imported dataset files, single commit for imports (#651) (75ce369)
- always add commit to dataset if possible (#648) (7659bc8), closes #646
- cleanup needed for integration tests on py35 (#653) (fdd7215)
- fixed serialization of datetime to iso format (#629) (693d59d)
- fixes broken integration test (#649) (04eba66)
- hide image, pull, runner, show, workon and deactivate commands (#672) (a3e9998)
- integration tests fixed (#685) (f0ea8f0)
- migration of old datasets (#639) (4d4d7d2)
- migration timezones (#683) (58c2de4)
- Removes unneccesary call to git lfs with no paths (#658) (e32d48b)
- renku home directory overwrite in tests (#657) (90e1c48)
- upload metadata before actual files (#652) (95ed468)
- use latest_html for version check (#647) (c6b0309), closes #641
- user-related metadata (#655) (44183e6)
- zenodo export failing with relative paths (d40967c)

#### Features

- dataverse import (#626) (9f0f9a1)
- enable all datasets command to operate on dirty repository (#607) (74e328b)
- explicit input output specification (#598) (ce8ba67)
- export filename as schema:name (#643) (aed54bf), closes #640
- support for indirect inputs and outputs (#650) (e960a98)

### 3.4.2 0.5.2 (2019-07-26)

#### Bug Fixes

- safe_path check always operates on str (#603) (7c1c34e)

#### Features

- add SoftwareAgent to Activity (#590) (a60c20c), closes #508

### 3.4.3 0.5.1 (2019-07-12)

#### Bug Fixes

- ensure external storage is handled correctly (#592) (7938ac4)
- only check local repo for lfs filter (#575) (a64dc79)
- **cli:** allow renku run with many inputs (f60783e), closes #552
- added check for overwriting datasets (#541) (8c697fb)
- escape whitespaces in notebook name (#584) (0542fcc)
- modify json-ld for datasets (#534) (ab6a719), closes #525 #526
- refactored tests and docs to align with updated pydoctstyle (#586) (6f981c8)
- **cli:** add check of missing references (9a373da)
- **cli:** fail when removing non existing dataset (dd728db)
- **status:** fix renku status output when not in root folder (#564) (873270d), closes #551
- added dependencies for SSL support (#565) (4fa0fed)
- **datasets:** strip query string from data filenames (450898b)
- fixed serialization of creators (#550) (6a9173c)
- updated docs (#539) (ff9a67c)
- **cli:** remove dataset aliases (6206e62)
- **cwl:** detect script as input parameter (e23b75a), closes #495
- **deps:** updated dependencies (691644d)

#### Features

- add dataset metadata to the KG (#558) (fb443d7)
- **datasets:** export dataset to zenodo (#529) (fc6fd4f)
- added support for working on dirty repo (ae67be7)
- **datasets:** edit dataset metadata (#549) (db39083)
- integrate metadata from zenodo (#545) (4273d2a)
- **config:** added global config manager (#533) (938f820)
- **datasets:** import data from zenodo (#509) (52b2769)

### 3.4.4 0.5.0 (2019-03-28)

#### Bug Fixes

- **api:** make methods lock free (1f63964), closes #486
- use safe_load for parsing yaml (5383d1e), closes #464
- **datasets:** link flag on dataset add (eae30f4)

#### Features

- **api:** list datasets from a commit (04a9fe9)
- **cli:** add dataset rm command (a70c7ce)
- **cli:** add rm command (cf0f502)
- **cli:** configurable format of dataset output (d37abf3)
- **dataset:** add existing file from current repo (575686b), closes #99
- **datasets:** added ls-files command (ccc4f59)
- **models:** reference context for relative paths (5d1e8e7), closes #452
- add JSON-LD output format for datasets (c755d7b), closes #426
- generate Makefile with log –format Makefile (1e440ce)

### 3.4.5 v0.4.0

*(released 2019-03-05)*

- Adds `renku mv` command which updates dataset metadata, `.gitattributes` and symlinks.
- Pulls LFS objects from submodules correctly.
- Adds listing of datasets.
- Adds reduced dot format for `renku log`.
- Adds `doctor` command to check missing files in datasets.
- Moves dataset metadata to `.renku/datasets` and adds `migrate datasets` command and uses UUID for metadata path.
- Gets git attrs for files to prevent duplicates in `.gitattributes`.
- Fixes `renku show outputs` for directories.
- Runs Git LFS checkout in a worktrees and lazily pulls necessary LFS files before running commands.
- Asks user before overriding an existing file using `renku init` or `renku runner template`.
- Fixes `renku init --force` in an empty dir.
- Renames `CommitMixin._location` to `_project`.
- Addresses issue with commits editing multiple CWL files.
- Exports merge commits for full lineage.
- Exports path and parent directories.
- Adds an automatic check for the latest version.
- Simplifies issue submission from traceback to GitHub or Sentry. Requires `SENTRY_DSN` variable to be set and *sentry-sdk* package to be installed before sending any data.
- Removes outputs before run.
- Allows update of directories.
- Improves readability of the status message.
- Checks ignored path when added to a dataset.

- Adds API method for finding ignored paths.
- Uses branches for `init --force`.
- Fixes CVE-2017-18342.
- Fixes regex for parsing Git remote URLs.
- Handles `--isolation` option using `git worktree`.
- Renames `client.git` to `client.repo`.
- Supports `python -m renku`.
- Allows '.' and '-' in repo path.

### 3.4.6 v0.3.3

*(released 2018-12-07)*

- Fixes generated Homebrew formula.
- Renames `renku pull path` to `renku storage pull` with deprecation warning.

### 3.4.7 v0.3.2

*(released 2018-11-29)*

- Fixes display of workflows in `renku log`.

### 3.4.8 v0.3.1

*(released 2018-11-29)*

- Fixes issues with parsing remote Git URLs.

### 3.4.9 v0.3.0

*(released 2018-11-26)*

- Adds JSON-LD context to objects extracted from the Git repository (see `renku show context --list`).
- Uses PROV-O and WFPROV as provenance vocabularies and generates "stable" object identifiers (`@id`) for RDF and JSON-LD output formats.
- Refactors the log output to allow linking files and directories.
- Adds support for aliasing tools and workflows.
- Adds option to install shell completion (`renku --install-completion`).
- Fixes initialization of Git submodules.
- Uses relative submodule paths when appropriate.
- Simplifies external storage configuration.

### 3.4.10 v0.2.0

*(released 2018-09-25)*

- Refactored version using Git and Common Workflow Language.

### 3.4.11 v0.1.0

*(released 2017-09-06)*

- Initial public release as Renga.

## 3.5 Glossary

**inputs** Files and/or directories which are required for running tools.

**outputs** Files and/or directories which are created or modified during an execution of a tool.

**tool** A description of a standalone, non-interactive program which can be invoked on some inputs, produces outputs, and then terminates[1].

[1] https://www.commonwl.org/v1.0/CommandLineTool.html

CHAPTER 4

# Full Table of Contents

## 4.1 How does this compare . . .

There are many tools that can be used for doing your day-to-day work. Renku is not **a silver bullet** or **a magic wand** for making your results reproducible.

### 4.1.1 ... to `Makefile`

If you are using `Makefile` to generate your *outputs* you are on a good path. However you might be missing versioning of your past executions.

Renku internally builds rules similar to those defined in a `Makefile` and makes sure that all files are saved before running a *tool*.

Running the following `renku run` commands

```
$ renku run echo test > foo
$ renku run wc -c < foo > foo.wc
```

is equivalent to this simple `Makefile`.

```
foo:
  @echo test > foo

foo.wc: foo
  @wc -c < foo > foo.wc
```

Renku also makes sure that if any of the *inputs* are modified only the necessary "rules" are invoked. In addition, *make* does not run the rule if all dependencies are older then the targets.

```
$ renku run echo second > foo
$ renku status
On branch master
```

(continues on next page)

(continued from previous page)

```
Files generated from newer inputs:
  (use "renku log [<file>...]" to see the full lineage)
  (use "renku update [<file>...]" to generate the file from its latest inputs)

      foo.wc: foo#deadbeef

$ renku update foo.wc
$ renku status
On branch master
All files were generated from the latest inputs.
```

**Note:** As a **bonus** the `Makefile` can be generated by running `renku log --format Makefile foo.wc` command.

## 4.2 Renku Command Line

The base command for interacting with the Renku platform.

### 4.2.1 `renku` (base command)

To list the available commands, either run `renku` with no parameters or execute `renku help`:

```
$ renku help
Usage: renku [OPTIONS] COMMAND [ARGS]...

Check common Renku commands used in various situations.


Options:
  --version                       Print version number.
  --config PATH                   Location of client config files.
  --config-path                   Print application config path.
  --install-completion            Install completion for the current shell.
  --path <path>                   Location of a Renku repository.
                                  [default: (dynamic)]
  --renku-home <path>             Location of the Renku directory.
                                  [default: .renku]
  --external-storage / -S, --no-external-storage
                                  Use an external file storage service.
  -h, --help                      Show this message and exit.

Commands:
  # [...]
```

#### Configuration files

Depending on your system, you may find the configuration files used by Renku command line in a different folder. By default, the following rules are used:

**MacOS:** `~/Library/Application Support/Renku`

**Unix:** `~/.config/renku`

**Windows:** `C:\Users\<user>\AppData\Roaming\Renku`

If in doubt where to look for the configuration file, you can display its path by running `renku --config-path`.

You can specify a different location via the `RENKU_CONFIG` environment variable or the `--config` command line option. If both are specified, then the `--config` option value is used. For example:

```
$ renku --config ~/renku/config/ init
```

instructs Renku to store the configuration files in your `~/renku/config/` directory when running the `init` command.

### 4.2.2 renku init

Create an empty Renku project or reinitialize an existing one.

#### Starting a Renku project

If you have an existing directory which you want to turn into a Renku project, you can type:

```
$ cd ~/my_project
$ renku init
```

or:

```
$ renku init ~/my_project
```

This creates a new subdirectory named `.renku` that contains all the necessary files for managing the project configuration.

If provided directory does not exist, it will be created.

#### Updating an existing project

There are situations when the required structure of a Renku project needs to be recreated or you have an **existing** Git repository. You can solve these situation by simply adding the `--force` option.

```
$ git init .
$ echo "# Example\nThis is a README." > README.md
$ git add README.md
$ git commit -m 'Example readme file'
# renku init would fail because there is a git repository
$ renku init --force
```

You can also enable the external storage system for output files, if it was not installed previously.

```
$ renku init --force --external-storage
```

### 4.2.3 renku config

Get and set Renku repository or global options.

#### Set values

You can set various Renku configuration options, for example the image registry URL, with a command like:

```
$ renku config registry https://registry.gitlab.com/demo/demo
```

#### Query values

You display a previously set value with:

```
$ renku config registry
https://registry.gitlab.com/demo/demo
```

### 4.2.4 `renku dataset`

Renku CLI commands for handling of datasets.

#### Manipulating datasets

Creating an empty dataset inside a Renku project:

```
$ renku dataset create my-dataset
Creating a dataset ... OK
```

Listing all datasets:

```
$ renku dataset
ID        NAME           CREATED              CREATORS
--------  -------------  -------------------  ---------
0ad1cb9a  some-dataset   2019-03-19 16:39:46  sam
9436e36c  my-dataset     2019-02-28 16:48:09  sam
```

Deleting a dataset:

```
$ renku dataset rm some-dataset
OK
```

#### Working with data

Adding data to the dataset:

```
$ renku dataset add my-dataset http://data-url
```

This will copy the contents of `data-url` to the dataset and add it to the dataset metadata.

To add data from a git repository, you can specify it via https or git+ssh URL schemes. For example,

```
$ renku dataset add my-dataset git+ssh://host.io/namespace/project.git
```

Sometimes you want to import just specific paths within the parent project. In this case, use the `--source` or `-s` flag:

```
$ renku dataset add my-dataset --source path/within/repo/to/datafile \
    git+ssh://host.io/namespace/project.git
```

The command above will result in a structure like

```
data/
  my-dataset/
    datafile
```

You can use `--destination` or `-d` flag to change the name of the target file or directory. The semantics here are similar to the POSIX copy command: if the destination does not exist or if it is a file then the source will be renamed; if the destination exists and is a directory the source will be copied to it. You will get an error message if you try to move a directory to a file or copy multiple files into one.

```
$ renku dataset add my-dataset \
    --source path/within/repo/to/datafile \
    --destination new-dir/new-filename \
    git+ssh://host.io/namespace/project.git
```

will yield:

```
data/
  my-dataset/
    new-dir/
      new-filename
```

Tagging a dataset:

A dataset can be tagged with an arbitrary tag to refer to the dataset at that point in time. A tag can be added like this:

```
$ renku dataset tag my-dataset 1.0 -d "Version 1.0 tag"
```

A list of all tags can be seen by running:

```
$ renku dataset ls-tags my-dataset
CREATED              NAME    DESCRIPTION      DATASET     COMMIT
-------------------  ------  ---------------  ----------  ----------------
2019-09-19 17:29:13  1.0     Version 1.0 tag  my-dataset  6c19a8d31545b...
```

A tag can be removed with:

```
$ renku dataset rm-tags my-dataset 1.0
```

Importing data from an external provider:

```
$ renku dataset import 10.5281/zenodo.3352150
```

This will import the dataset with the DOI (Digital Object Identifier) `10.5281/zenodo.3352150` and make it locally available. Dataverse and Zenodo are supported, with DOIs (e.g. `10.5281/zenodo.3352150` or `doi:10.5281/zenodo.3352150`) and full URLs (e.g. `http://zenodo.org/record/3352150`). A tag with the remote version of the dataset is automatically created.

Exporting data to an external provider:

```
$ renku dataset export my-dataset zenodo
```

This will export the dataset `my-dataset` to `zenodo.org` as a draft, allowing for publication later on. If the dataset has any tags set, you can chose if the repository *HEAD* version or one of the tags should be exported. The remote version will be set to the local tag that is being exported.

Listing all files in the project associated with a dataset.

```
$ renku dataset ls-files
ADDED                CREATORS    DATASET         PATH
-------------------  ---------  -------------  --------------------------
2019-02-28 16:48:09  sam        my-dataset     ...my-dataset/addme
2019-02-28 16:49:02  sam        my-dataset     ...my-dataset/weather/file1
2019-02-28 16:49:02  sam        my-dataset     ...my-dataset/weather/file2
2019-02-28 16:49:02  sam        my-dataset     ...my-dataset/weather/file3
```

Sometimes you want to filter the files. For this we use `--dataset`, `--include` and `--exclude` flags:

```
$ renku dataset ls-files --include "file*" --exclude "file3"
ADDED                CREATORS    DATASET     PATH
-------------------  ---------  ----------  ----------------------------
2019-02-28 16:49:02  sam        my-dataset  .../my-dataset/weather/file1
2019-02-28 16:49:02  sam        my-dataset  .../my-dataset/weather/file2
```

Unlink a file from a dataset:

```
$ renku dataset unlink my-dataset --include file1
OK
```

Unlink all files within a directory from a dataset:

```
$ renku dataset unlink my-dataset --include "weather/*"
OK
```

Unlink all files from a dataset:

```
$ renku dataset unlink my-dataset
Warning: You are about to remove following from "my-dataset" dataset.
.../my-dataset/weather/file1
.../my-dataset/weather/file2
.../my-dataset/weather/file3
Do you wish to continue? [y/N]:
```

**Note:** The `unlink` command does not delete files, only the dataset record.

### 4.2.5 `renku run`

Track provenance of data created by executing programs.

#### Capture command line execution

Tracking execution of your command line script is done by simply adding the `renku run` command before the actual command. This will enable detection of:

- arguments (flags),
- string and integer options,

- input files or directories if linked to existing paths in the repository,
- output files or directories if modified or created while running the command.

**Note:** If there were uncommitted changes in the repository, then the `renku run` command fails. See **`git status`** for details.

**Warning:** Input and output paths can only be detected if they are passed as arguments to `renku run`.

### Detecting input paths

Any path passed as an argument to `renku run`, which was not changed during the execution, is identified as an input path. The identification only works if the path associated with the argument matches an existing file or directory in the repository.

The detection might not work as expected if:

- a file is **modified** during the execution. In this case it will be stored as an **output**;
- a path is not passed as an argument to `renku run`.

**Specifying auxiliary inputs (`--input`)**

You can specify extra inputs to your program explicitly by using the `--input` option. This is useful for specifying hidden dependencies that don't appear on the command line. These input file must exist before execution of `renku run` command. This option is not a replacement for the arguments that are passed on the command line. Files or directories specified with this option will not be passed as input arguments to the script.

### Detecting output paths

Any path **modified** or **created** during the execution will be added as an output.

Because the output path detection is based on the Git repository state after the execution of `renku run` command, it is good to have a basic understanding of the underlying principles and limitations of tracking files in Git.

Git tracks not only the paths in a repository, but also the content stored in those paths. Therefore:

- a recreated file with the same content is not considered an output file, but instead is kept as an input;
- file moves are detected based on their content and can cause problems;
- directories cannot be empty.

**Note:** When in doubt whether the outputs will be detected, remove all outputs using `git rm <path>` followed by `git commit` before running the `renku run` command.

**Command does not produce any files (`--no-output`)**

If the program does not produce any outputs, the execution ends with an error:

```
Error: There are not any detected outputs in the repository.
```

You can specify the `--no-output` option to force tracking of such an execution.

**Specifying outputs explicitly (`--output`)**

You can specify expected outputs of your program explicitly by using the `--output` option. These output must exist after the execution of the `renku run` command. However, they do not need to be modified by the command.

### Detecting standard streams

Often the program expect inputs as a standard input stream. This is detected and recorded in the tool specification when invoked by `renku run cat < A`.

Similarly, both redirects to standard output and standard error output can be done when invoking a command:

```
$ renku run grep "test" B > C 2> D
```

**Warning:** Detecting inputs and outputs from pipes `|` is not supported.

### Specifying inputs and outputs programmatically

Sometimes the list of inputs and outputs are not known before execution of the program. For example, a program might accept a date range as input and access all files within that range during its execution.

To address this issue, the program can dump a list of input and output files that it is accessing in `inputs.txt` and `outputs.txt`. Each line in these files is expected to be the path to an input or output file within the project's directory. When the program is finished, Renku will look for existence of these two files and adds their content to the list of explicit inputs and outputs. Renku will then delete these two files.

By default, Renku looks for these two files in `.renku/tmp` directory. One can change this default location by setting `RENKU_FILELIST_PATH` environment variable. When set, it points to the directory within the project's directory where `inputs.txt` and `outputs.txt` reside.

### Exit codes

All Unix commands return a number between 0 and 255 which is called "exit code". In case other numbers are returned, they are treaded module 256 (-10 is equivalent to 246, 257 is equivalent to 1). The exit-code 0 represents a *success* and non-zero exit-code indicates a *failure*.

Therefore the command specified after `renku run` is expected to return exit-code 0. If the command returns different exit code, you can specify them with `--success-code=<INT>` parameter.

```
$ renku run --success-code=1 --no-output fail
```

## 4.2.6 `renku log`

Show provenance of data created by executing programs.

#### File provenance

Unlike the traditional file history format, which shows previous revisions of the file, this format presents tool inputs together with their revision identifiers.

A `*` character shows to which lineage the specific file belongs to. A `@` character in the graph lineage means that the corresponding file does not have any inputs and the history starts there.

When called without file names, `renku log` shows the history of most recently created files. With the `--revision <refname>` option the output is shown as it was in the specified revision.

#### Provenance examples

**`renku log B`** Show the history of file `B` since its last creation or modification.

**`renku log --revision HEAD~5`** Show the history of files that have been created or modified 5 commits ago.

**`renku log --revision e3f0bd5a D E`** Show the history of files `D` and `E` as it looked in the commit `e3f0bd5a`.

#### Output formats

Following formats supported when specified with `--format` option:

- *ascii*
- *dot*

You can generate a PNG of the full history of all files in the repository using the **`dot`** program.

```
$ FILES=$(git ls-files --no-empty-directory --recurse-submodules)
$ renku log --format dot $FILES | dot -Tpng > /tmp/graph.png
$ open /tmp/graph.png
```

### 4.2.7 `renku status`

Show status of data files created in the repository.

#### Inspecting a repository

Displays paths of outputs which were generated from newer inputs files and paths of files that have been used in diverent versions.

The first paths are what need to be recreated by running `renku update`. See more in section about *renku update*.

The paths mentioned in the output are made relative to the current directory if you are working in a subdirectory (this is on purpose, to help cutting and pasting to other commands). They also contain first 8 characters of the corresponding commit identifier after the `#` (hash). If the file was imported from another repository, the short name of is shown together with the filename before `@`.

### 4.2.8 `renku update`

Update outdated files created by the "run" command.

### Recreating outdated files

The information about dependencies for each file in the repository is generated from information stored in the underlying Git repository.

A minimal dependency graph is generated for each outdated file stored in the repository. It means that only the necessary steps will be executed and the workflow used to orchestrate these steps is stored in the repository.

Assume that the following history for the file `H` exists.

```
      C---D---E
     /         \
A---B---F---G---H
```

The first example shows situation when `D` is modified and files `E` and `H` become outdated.

```
      C--*D*--(E)
     /           \
A---B---F---G---(H)

** - modified
() - needs update
```

In this situation, you can do efectively two things:

- Recreate a single file by running

```
$ renku update E
```

- Update all files by simply running

```
$ renku update
```

**Note:** If there were uncommitted changes then the command fails. Check **`git status`** to see details.

### Pre-update checks

In the next example, files `A` or `B` are modified, hence the majority of dependent files must be recreated.

```
        (C)--(D)--(E)
       /             \
*A*--*B*--(F)--(G)--(H)
```

To avoid excesive recreation of the large portion of files which could have been affected by a simple change of an input file, consider speficing a single file (e.g. `renku update G`). See also *renku status*.

### Update siblings

If a tool produces multiple output files, these outputs need to be always updated together.

```
               (B)
              /
*A*--[step 1]--(C)
              \
               (D)
```

An attempt to update a single file would fail with the following error.

```
$ renku update C
Error: There are missing output siblings:

     B
     D

Include the files above in the command or use --with-siblings option.
```

The following commands will produce the same result.

```
$ renku update --with-siblings C
$ renku update B C D
```

### 4.2.9 renku rerun

Recreate files created by the "run" command.

#### Recreating files

Assume you have run a step 2 that uses a stochastic algorithm, so each run will be slightly different. The goal is to regenerate output `C` several times to compare the output. In this situation it is not possible to simply call *renku update* since the input file `A` has not been modified after the execution of step 2.

```
A-[step 1]-B-[step 2*]-C
```

Recreate a specific output file by running:

```
$ renku rerun C
```

If you would like to recreate a file which was one of several produced by a tool, then these files must be recreated as well. See the explanation in *updating siblings*.

### 4.2.10 renku rm

Remove a file, a directory, or a symlink.

Removing a file that belongs to a dataset will update its metadata. It also will attempt to update tracking information for files stored in an external storage (using Git LFS).

### 4.2.11 renku mv

Move or rename a file, a directory, or a symlink.

Moving a file that belongs to a dataset will update its metadata. It also will attempt to update tracking information for files stored in an external storage (using Git LFS). Finally it makes sure that all relative symlinks work after the move.

### 4.2.12 renku workflow

Manage the set of CWL files created by `renku` commands.

With no arguments, shows a list of captured CWL files. Several subcommands are available to perform operations on CWL files.

#### Reference tools and workflows

Managing large number of tools and workflows with automatically generated names may be cumbersome. The names can be added to the last executed `run`, `rerun` or `update` command by running `renku workflow set-name <name>`. The name can be added to an arbitrary file in `.renku/workflow/*.cwl` anytime later.

### 4.2.13 `renku show`

Show information about objects in current repository.

#### Siblings

In situations when multiple outputs have been generated by a single `renku run` command, the siblings can be discovered by running `renku show siblings PATH` command.

Assume that the following graph represents relations in the repository.

```
      D---E---G
     /     \
A---B---C   F
```

Then the following outputs would be shown.

```
$ renku show siblings C
C
D
$ renku show siblings G
F
G
$ renku show siblings A
A
```

#### Input and output files

You can list input and output files generated in the repository by running `renku show inputs` and `renku show outputs` commands. Alternatively, you can check if all paths specified as arguments are input or output files respectively.

```
$ renku run wc < source.txt > result.wc
$ renku show inputs
source.txt
$ renku show outputs
result.wc
$ renku show outputs source.txt
$ echo $?  # last command finished with an error code
1
```

### 4.2.14 renku storage

Manage an external storage.

### 4.2.15 renku doctor

Check your system and repository for potential problems.

### 4.2.16 renku migrate

Migrate files and metadata to the latest Renku version.

#### Datasets

The location of dataset metadata files has been changed from the `data/<name>/metadata.yml` to `.renku/datasets/<UUID>/metadata.yml`. All file paths inside a metadata file are relative to itself and the `renku migrate datasets` command will take care of it.

### 4.2.17 renku githooks

Install and uninstall Git hooks.

#### Prevent modifications of output files

The commit hooks are enabled by default to prevent situation when some output file is manually modified.

```
$ renku init
$ renku run echo hello > greeting.txt
$ edit greeting.txt
$ git commit greeting.txt
You are trying to update some output files.

Modified outputs:
  greeting.txt

If you are sure, use "git commit --no-verify".
```

### 4.2.18 Error Tracking

Renku is not bug-free and you can help us to find them.

#### GitHub

You can quickly open an issue on GitHub with a traceback and minimal system information when you hit an unhandled exception in the CLI.

```
Ahhhhhhhh! You have found a bug.

1. Open an issue by typing "open";
2. Print human-readable information by typing "print";
3. See the full traceback without submitting details (default: "ignore").

Please select an action by typing its name (open, print, ignore) [ignore]:
```

#### Sentry

When using `renku` as a hosted service the Sentry integration can be enabled to help developers iterate faster by showing them where bugs happen, how often, and who is affected.

1. Install Sentry-SDK with `python -m pip install sentry-sdk`;
2. Set environment variable `SENTRY_DSN=https://<key>@sentry.<domain>/<project>`.

**Warning:** User information might be sent to help resolving the problem. If you are not using your own Sentry instance you should inform users that you are sending possibly sensitive information to a 3rd-party service.

## 4.3 Models

Model objects used in Python SDK.

### 4.3.1 Projects

Model objects representing projects.

**class** renku.core.models.projects.**Project**(*name=None*, *created=NOTHING*, *updated=NOTHING*, *version='2'*, ***, *client=None*, *creator=None*, *id=None*)

Represent a project.

**Type:**

```
["prov:Location", "schema:Project"]
```

**Context:**

```
{
  "schema": "http://schema.org/",
  "prov": "http://www.w3.org/ns/prov#",
  "name": "schema:name",
  "created": "schema:dateCreated",
  "updated": "schema:dateUpdated",
  "version": "schema:schemaVersion",
  "creator": "schema:creator",
  "_id": "@id"
}
```

**class** renku.core.models.projects.**ProjectCollection**(*client=None*)

Represent projects on the server.

**Example**

Create a project and check its name.

# >>> project = client.projects.create(name='test-project') # >>> project.name # 'test-project'

Create a representation of objects on the server.

**class Meta**
Information about individual projects.

**model**
alias of *Project*

**create**(*name=None*, ***kwargs*)
Create a new project.

**Parameters** **name** – The name of the project.

**Returns** An instance of the newly create project.

**Return type** *renku.core.models.projects.Project*

### 4.3.2 Datasets

Model objects representing datasets.

#### Dataset object

**class** renku.core.models.datasets.**Dataset**(***, *commit=None*, *client=None*, *path=None*, *project=None*, *parent=None*, *id=None*, *label=None*, *creator=NOTHING*, *date_published=None*, *description=''*, *identifier=NOTHING*, *in_language=None*, *keywords=NOTHING*, *based_on=None*, *license=None*, *name: str = None*, *url=None*, *version=None*, *created=NOTHING*, *files=NOTHING*, *tags=NOTHING*, *same_as=None*)
Repesent a dataset.

**Type:**

```
["prov:Entity", "schema:Dataset", "wfprov:Artifact"]
```

**Context:**

```
{
  "affiliation": "schema:affiliation",
  "alternate_name": "schema:alternateName",
  "email": "schema:email",
  "schema": "http://schema.org/",
  "prov": "http://www.w3.org/ns/prov#",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
```

(continues on next page)

(continued from previous page)

```
  "creator": "schema:creator",
  "date_published": "schema:datePublished",
  "description": "schema:description",
  "identifier": "schema:identifier",
  "in_language": "schema:inLanguage",
  "keywords": "schema:keywords",
  "based_on": "schema:isBasedOn",
  "license": "schema:license",
  "name": "schema:name",
  "url": "schema:url",
  "version": "schema:version",
  "created": "schema:dateCreated",
  "files": "schema:hasPart",
  "tags": "schema:subjectOf",
  "same_as": "schema:sameAs"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**creators_csv**
Comma-separated list of creators associated with dataset.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_reference**()
Create a default reference path.

**display_name**
Get dataset display name.

**editable**
Subset of attributes which user can edit.

**entities**
Yield itself.

**find_file**(*filename*, *return_index=False*)
Find a file in files container.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_revision**(*client*, *path*, *revision='HEAD'*, *parent=None*, ***kwargs*)
Return dependency from given path and revision.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**parent**
Return the parent object.

**rename_files**(*rename*)
Rename files using the path mapping function.

**short_id**
Shorter version of identifier.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**uid**
UUID part of identifier.

**unlink_file**(*file_path*)
Unlink a file from dataset.

**Parameters** **file_path** – Relative path used as key inside files container.

**update_files**(*files*)
Update files with collection of DatasetFile objects.

**update_metadata**(*other_dataset*)
Updates instance attributes with other dataset attributes.

**Parameters** **other_dataset** – *Dataset*

**Returns** self

### Dataset file

Manage files in the dataset.

**class** renku.core.models.datasets.**DatasetFile**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=None*, *parent=None*, *creator=NOTHING*, *added=NOTHING*, *checksum=None*, *dataset=None*, *filename=NOTHING*, *name=None*, *filesize=None*, *filetype=None*, *url=None*, *based_on=None*)

Represent a file in a dataset.

**Type:**

```
["prov:Entity", "schema:DigitalDocument", "wfprov:Artifact"]
```

**Context:**

```
{
  "schema": "http://schema.org/",
  "prov": "http://www.w3.org/ns/prov#",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
  "creator": "schema:creator",
  "added": "schema:dateCreated",
  "dataset": "schema:isPartOf",
  "name": "schema:name",
  "url": "schema:url",
```

(continues on next page)

(continued from previous page)

```
  "based_on": "schema:isBasedOn"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**creators_csv**
Comma-separated list of creators associated with dataset.

**default_filename**()
Generate default filename based on path.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_reference**()
Create a default reference path.

**entities**
Yield itself.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_revision**(*client*, *path*, *revision='HEAD'*, *parent=None*, ***kwargs*)
Return dependency from given path and revision.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**full_path**
Return full path in the current reference frame.

**parent**
Return the parent object.

**size_in_mb**
Return file size in megabytes.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

### Creator

**class** renku.core.models.datasets.**Creator**(***, *client=None*, *affiliation=None*, *email=None*, *alternate_name=None*, *name=None*, *id=NOTHING*)
Represent the creator of a resource.

**Type:**

```
"schema:Person"
```

**Context:**

```
{
  "schema": "http://schema.org/",
  "affiliation": "schema:affiliation",
  "email": "schema:email",
  "alternate_name": "schema:alternateName",
  "name": "schema:name",
  "_id": "@id"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**check_email**(*attribute*, *value*)
Check that the email is valid.

**default_id**()
Set the default id.

**default_reference**()
Create a default reference path.

**classmethod from_commit**(*commit*)
Create an instance from a Git commit.

**classmethod from_git**(*git*)
Create an instance from a Git repo.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**short_name**
Gives full name in short form.

**to_yaml**()
Store an instance to the referenced YAML file.

### 4.3.3 Provenance

Extract provenance information from the repository.

### Activities

**class** renku.core.models.provenance.activities.**Activity**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=None*, *id=NOTHING*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *process=None*, *outputs=NOTHING*, *generated=NOTHING*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agent=SoftwareAgent(label='renku 0.6.1'*, *was_started_by=None*, *_id='https://github.com/swissdatasciencecenter/renk python/tree/v0.6.1')*, *person_agent=NOTHING*)

Represent an activity in the repository.

**Type:**

```
"prov:Activity"
```

**Context:**

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "schema": "http://schema.org/",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
  "_id": "@id",
  "_message": "rdfs:comment",
  "_was_informed_by": "prov:wasInformedBy",
  "generated": {
    "@reverse": "prov:activity"
  },
  "influenced": "prov:influenced",
  "started_at_time": {
    "@id": "prov:startedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "ended_at_time": {
    "@id": "prov:endedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "agent": "prov:agent",
  "person_agent": "prov:agent"
}
```

**asjsonld**()

Create JSON-LD with the original source data.

**default_ended_at_time**()
Configure calculated properties.

**default_generated**()
Entities generated by this Action.

**default_id**()
Configure calculated ID.

**default_influenced**()
Calculate default values.

**default_label**()
Generate a default label.

**default_message**()
Generate a default message.

**default_outputs**()
Guess default outputs from a commit.

**default_person_agent**()
Set person agent to be the author of the commit.

**default_reference**()
Create a default reference path.

**default_started_at_time**()
Configure calculated properties.

**default_was_informed_by**()
List parent actions.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**classmethod generate_id**(*commit*)
Calculate action ID.

**nodes**
Return topologically sorted nodes.

**parents**
Return parent commits.

**paths**
Return all paths in the commit.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.activities.**ProcessRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=None*, *id=NOTHING*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *process=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agent=SoftwareAgent(label='renku 0.6.1'*, *was_started_by=None*, *_id='https://github.com/swissdatasciencecenter/re python/tree/v0.6.1')*, *person_agent=NOTHING*, *inputs=NOTHING*, *outputs=NOTHING*, *generated=NOTHING*, *association=None*, *qualified_usage=NOTHING*)

A process run is a particular execution of a Process description.

**Type:**

```
["prov:Activity", "wfprov:ProcessRun"]
```

**Context:**

```
{
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "prov": "http://www.w3.org/ns/prov#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "schema": "http://schema.org/",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
  "_id": "@id",
  "_message": "rdfs:comment",
  "_was_informed_by": "prov:wasInformedBy",
  "generated": {
    "@reverse": "prov:activity"
  },
  "influenced": "prov:influenced",
  "started_at_time": {
    "@id": "prov:startedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "ended_at_time": {
    "@id": "prov:endedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "agent": "prov:agent",
  "person_agent": "prov:agent",
```

(continues on next page)

(continued from previous page)

```
  "association": "prov:qualifiedAssociation",
  "qualified_usage": "prov:qualifiedUsage"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_ended_at_time**()
Configure calculated properties.

**default_generated**()
Calculate default values.

**default_id**()
Configure calculated ID.

**default_influenced**()
Calculate default values.

**default_inputs**()
Guess default inputs from a process.

**default_label**()
Generate a default label.

**default_message**()
Generate a default message.

**default_outputs**()
Guess default outputs from a process.

**default_person_agent**()
Set person agent to be the author of the commit.

**default_qualified_usage**()
Generate list of used artifacts.

**default_reference**()
Create a default reference path.

**default_started_at_time**()
Configure calculated properties.

**default_was_informed_by**()
List parent actions.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**classmethod generate_id**(*commit*)
Calculate action ID.

**iter_output_files**(*commit=None*)
Yield tuples with output id and path.

**nodes**
Return topologically sorted nodes.

**parents**
Return parent commits.

**paths**
Return all paths in the commit.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.activities.**WorkflowRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=None*, *id=NOTHING*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *process=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agent=SoftwareAgent(label='renku 0.6.1'*, *was_started_by=None*, *_id='https://github.com/swissdatasciencecenter/python/tree/v0.6.1')*, *person_agent=NOTHING*, *inputs=NOTHING*, *association=None*, *qualified_usage=NOTHING*, *children=NOTHING*, *processes=NOTHING*, *subprocesses=NOTHING*, *outputs=NOTHING*, *generated=NOTHING*)

A workflow run typically contains several subprocesses.

**Type:**

```
["prov:Activity", "wfprov:ProcessRun", "wfprov:WorkflowRun"]
```

**Context:**

```
{
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "prov": "http://www.w3.org/ns/prov#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "schema": "http://schema.org/",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
```

(continues on next page)

(continued from previous page)

```
  "_id": "@id",
  "_message": "rdfs:comment",
  "_was_informed_by": "prov:wasInformedBy",
  "generated": {
    "@reverse": "prov:activity"
  },
  "influenced": "prov:influenced",
  "started_at_time": {
    "@id": "prov:startedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "ended_at_time": {
    "@id": "prov:endedAtTime",
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
  },
  "agent": "prov:agent",
  "person_agent": "prov:agent",
  "association": "prov:qualifiedAssociation",
  "qualified_usage": "prov:qualifiedUsage",
  "_processes": {
    "@reverse": "wfprov:wasPartOfWorkflowRun"
  }
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_children**()
Load children from process.

**default_ended_at_time**()
Configure calculated properties.

**default_generated**()
Calculate default values.

**default_id**()
Configure calculated ID.

**default_influenced**()
Calculate default values.

**default_inputs**()
Guess default inputs from a process.

**default_label**()
Generate a default label.

**default_message**()
Generate a default message.

**default_outputs**()
Guess default outputs from a workflow.

**default_person_agent**()
Set person agent to be the author of the commit.

**default_qualified_usage**()
Generate list of used artifacts.

**default_reference**()
Create a default reference path.

**default_started_at_time**()
Configure calculated properties.

**default_subprocesses**()
Load subprocesses.

**default_was_informed_by**()
List parent actions.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**classmethod generate_id**(*commit*)
Calculate action ID.

**iter_output_files**(*commit=None*)
Yield tuples with output id and path.

**nodes**
Yield all graph nodes.

**parents**
Return parent commits.

**paths**
Return all paths in the commit.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

### Entities and Plans

**class** renku.core.models.provenance.entities.**Entity**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=None*, *parent=None*)

Represent a data value or item.

**Type:**

```
["prov:Entity", "wfprov:Artifact"]
```

**Context:**

```
{
  "schema": "http://schema.org/",
  "prov": "http://www.w3.org/ns/prov#",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
```

(continues on next page)

(continued from previous page)

```
  "_label": "rdfs:label",
  "_project": "schema:isPartOf"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_reference**()
Create a default reference path.

**entities**
Yield itself.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_revision**(*client*, *path*, *revision='HEAD'*, *parent=None*, ***kwargs*)
Return dependency from given path and revision.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**parent**
Return the parent object.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.entities.**Collection**(***, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=None*, *parent=None*, *members=NOTHING*)
Represent a directory with files.

**Type:**

```
["prov:Collection", "prov:Entity", "wfprov:Artifact"]
```

**Context:**

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "schema": "http://schema.org/",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
```

(continues on next page)

(continued from previous page)

```
  "_project": "schema:isPartOf",
  "members": "prov:hadMember"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_members**()
Generate default members as entities from current path.

**default_reference**()
Create a default reference path.

**entities**
Recursively return all files.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_revision**(*client*, *path*, *revision='HEAD'*, *parent=None*, ***kwargs*)
Return dependency from given path and revision.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**parent**
Return the parent object.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.entities.**Process**(***, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=None*, *activity*)
Represent a process.

**Type:**

```
["prov:Entity", "prov:Plan", "wfdesc:Process"]
```

**Context:**

```
{
  "wfdesc": "http://purl.org/wf4ever/wfdesc#",
  "prov": "http://www.w3.org/ns/prov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
```

(continues on next page)

(continued from previous page)

```
  "_activity": "prov:activity"
}
```

**activity**
Return the activity object.

**asjsonld**()
Create JSON-LD with the original source data.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_reference**()
Create a default reference path.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.entities.**Workflow**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=None*, *activity*, *subprocesses=NOTHING*)
Represent workflow with subprocesses.

**Type:**

```
["prov:Entity", "prov:Plan", "wfdesc:Process", "wfdesc:Workflow"]
```

**Context:**

```
{
  "wfdesc": "http://purl.org/wf4ever/wfdesc#",
  "prov": "http://www.w3.org/ns/prov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "schema:isPartOf",
  "_activity": "prov:activity",
  "subprocesses": "wfdesc:hasSubProcess"
}
```

**activity**
Return the activity object.

**asjsonld**()
Create JSON-LD with the original source data.

**default_id**()
Configure calculated ID.

**default_label**()
Generate a default label.

**default_reference**()
Create a default reference path.

**default_subprocesses**()
Load subprocesses.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**submodules**
Proxy to client submodules.

**to_yaml**()
Store an instance to the referenced YAML file.

### Agents

**class** renku.core.models.provenance.agents.**Person**(*name*, *email*)
Represent a person.

**Type:**

```
["prov:Person", "schema:Person"]
```

**Context:**

```
{
  "schema": "http://schema.org/",
  "prov": "http://www.w3.org/ns/prov#",
  "name": "rdfs:label",
  "email": "schema:email",
  "_id": "@id"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**check_email**(*attribute*, *value*)
Check that the email is valid.

**default_id**()
Configure calculated ID.

**default_reference**()
Create a default reference path.

**classmethod from_commit**(*commit*)
Create an instance from a Git commit.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.agents.**SoftwareAgent**(*, *label*, *was_started_by=None*, *id*)
Represent executed software.

**Type:**

```
["prov:SoftwareAgent", "wfprov:WorkflowEngine"]
```

**Context:**

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "label": "rdfs:label",
  "was_started_by": "prov:wasStartedBy",
  "_id": "@id"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_reference**()
Create a default reference path.

**classmethod from_commit**(*commit*)
Create an instance from a Git commit.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**to_yaml**()
Store an instance to the referenced YAML file.

### Relations

**class** renku.core.models.provenance.qualified.**Usage**(*, *entity*, *role=None*, *id=None*)
Represent a dependent path.

**Type:**

```
"prov:Usage"
```

**Context:**

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "entity": "prov:entity",
  "role": "prov:hadRole",
  "_id": "@id"
}
```

**asjsonld**()
Create JSON-LD with the original source data.

**default_reference**()
Create a default reference path.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_revision**(*client*, *path*, *revision='HEAD'*, ***kwargs*)
Return dependency from given path and revision.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**to_yaml**()
Store an instance to the referenced YAML file.

**class** renku.core.models.provenance.qualified.**Generation**(*entity*, *role=None*, ***, *activity=None*, *id=NOTHING*)
Represent an act of generating a file.

**Type:**

```
"prov:Generation"
```

**Context:**

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "entity": {
    "@reverse": "prov:qualifiedGeneration"
  },
  "role": "prov:hadRole",
  "_id": "@id"
}
```

**activity**
Return the activity object.

**asjsonld**()
Create JSON-LD with the original source data.

**default_id**()
Configure calculated ID.

**default_reference**()
Create a default reference path.

**classmethod from_jsonld**(*data*, *client=None*, *commit=None*, *__reference__=None*, *__source__=None*)
Instantiate a JSON-LD class from data.

**classmethod from_yaml**(*path*, *client=None*, *commit=None*)
Return an instance from a YAML file.

**to_yaml**()
Store an instance to the referenced YAML file.

### 4.3.4 Tools and Workflows

Manage creation of tools and workflows using the Common Workflow Language (CWL).

#### Common Workflow language

Renku uses CWL to represent runnable steps (tools) along with their inputs and outputs. Similarly, tools can be chained together to form CWL-defined workflows.

#### Command-line tool

Represent a `CommandLineTool` from the Common Workflow Language.

**class** renku.core.models.cwl.command_line_tool.**CommandLineTool**(*requirements=NOTHING*, *hints=NOTHING*, *label=None*, *doc=None*, *cwlVersion='v1.0'*, *baseCommand=''*, *arguments=NOTHING*, *stdin=None*, *stdout=None*, *stderr=None*, *inputs=NOTHING*, *outputs=NOTHING*, *successCodes=NOTHING*, *temporaryFailCodes=NOTHING*, *permanentFailCodes=NOTHING*)
Represent a command line tool.

**STD_STREAMS_REPR = {'stderr': '2>', 'stdin': '<', 'stdout': '>'}**
Format streams for a shell command representation.

**create_run**(***kwargs*)
Return an instance of process run.

**get_output_id**(*path*)
Return an id of the matching path from default values.

**to_argv**(*job=None*)
Generate arguments for system call.

**class** renku.core.models.cwl.command_line_tool.**CommandLineToolFactory**(*command_line*, *explicit_inputs=[]*, *explicit_outputs=[]*, *directory='.'*, *working_dir='.'*, *stdin=None*, *stderr=None*, *stdout=None*, *successCodes=NOTHING*)

Command Line Tool Factory.

**add_indirect_inputs**()
Read indirect inputs list and add them to explicit inputs.

**add_indirect_outputs**()
Read indirect outputs list and add them to explicit outputs.

**delete_indirect_files_list**()
Remove indirect inputs and outputs list.

**file_candidate**(*candidate*, *ignore=None*)
Return a path instance if it exists in current directory.

**find_explicit_inputs**()
Yield explicit inputs and command line input bindings if any.

**find_explicit_outputs**(*starting_output_id*)
Yield explicit output and changed command input parameter.

**generate_tool**()
Return an instance of command line tool.

**guess_inputs**(**arguments*)
Yield command input parameters and command line bindings.

**guess_outputs**(*paths*)
Yield detected output and changed command input parameter.

**guess_type**(*value*, *ignore_filenames=None*)
Return new value and CWL parameter type.

**read_files_list**(*files_list*)
Read files list where each line is a filepath.

**split_command_and_args**()
Return tuple with command and args from command line arguments.

**validate_command_line**(*attribute*, *value*)
Check the command line structure.

**validate_path**(*attribute*, *value*)
Path must exists.

**watch**(*client*, *no_output=False*)
Watch a Renku repository for changes to detect outputs.

renku.core.models.cwl.command_line_tool.**convert_arguments**(*value*)
Convert arguments from various input formats.

### Parameter

Represent parameters from the Common Workflow Language.

**class** renku.core.models.cwl.parameter.**CommandInputParameter**(*id=None*, *streamable=None*, *type='string'*, *description=None*, *default=None*, *inputBinding=None*)
An input parameter for a CommandLineTool.

**classmethod from_cwl**(*data*)
Create instance from type definition.

**to_argv**(***kwargs*)
Format command input parameter as shell argument.

**class** renku.core.models.cwl.parameter.**CommandLineBinding**(*position=None*, *prefix=None*, *separate: bool = True*, *itemSeparator=None*, *valueFrom=None*, *shellQuote: bool = True*)
Define the binding behavior when building the command line.

**to_argv**(*default=None*)
Format command line binding as shell argument.

**class** renku.core.models.cwl.parameter.**CommandOutputBinding**(*glob=None*)
Define the binding behavior for outputs.

**class** renku.core.models.cwl.parameter.**CommandOutputParameter**(*id=None*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*)
Define an output parameter for a CommandLineTool.

**class** renku.core.models.cwl.parameter.**InputParameter**(*id=None*, *streamable=None*, *type='string'*, *description=None*, *default=None*, *inputBinding=None*)
An input parameter.

**class** renku.core.models.cwl.parameter.**OutputParameter**(*id=None*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*)
An output parameter.

**class** renku.core.models.cwl.parameter.**Parameter**(*streamable=None*)
Define an input or output parameter to a process.

**class** renku.core.models.cwl.parameter.**WorkflowOutputParameter**(*id=None*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*, *outputSource=None*)

Define an output parameter for a Workflow.

renku.core.models.cwl.parameter.**convert_default**(*value*)

Convert a default value.

### Process

Represent a `Process` from the Common Workflow Language.

**class** renku.core.models.cwl.process.**Process**

Represent a process.

**iter_input_files**(*basedir*)

Yield tuples with input id and path.

### Types

Represent the Common Workflow Language types.

**class** renku.core.models.cwl.types.**Directory**(*path=None*, *listing=NOTHING*)

Represent a directory.

**class** renku.core.models.cwl.types.**Dirent**(*entryname=None*, *entry=None*, *writable=False*)

Define a file or subdirectory.

**class** renku.core.models.cwl.types.**File**(*path*)

Represent a file.

**class** renku.core.models.cwl.types.**PathFormatterMixin**

Format path property.

### Workflow

Represent workflows from the Common Workflow Language.

**class** renku.core.models.cwl.workflow.**Workflow**(*inputs=NOTHING*, *requirements=NOTHING*, *hints=NOTHING*, *label=None*, *doc=None*, *cwlVersion='v1.0'*, *outputs=NOTHING*, *steps=NOTHING*)

Define a workflow representation.

**add_step**(***kwargs*)

Add a workflow step.

**create_run**(***kwargs*)

Return an instance of process run.

**get_output_id**(*path*)
Return an id of the matching path from default values.

**topological_steps**
Return topologically sorted steps.

**class** renku.core.models.cwl.workflow.**WorkflowStep**(*run*, *id=NOTHING*, *in_=None*, *out=None*)
Define an executable element of a workflow.

renku.core.models.cwl.workflow.**convert_run**(*value*)
Convert value to CWLClass if dict is given.

### 4.3.5 File References

Manage names of Renku objects.

**class** renku.core.models.refs.**LinkReference**(*client*, *name*)
Manage linked object names.

**REFS = 'refs'**
Define a name of the folder with references in the Renku folder.

**classmethod check_ref_format**(*name*)
Ensures that a reference name is well formed.

It follows Git naming convention:

- any path component of it begins with ".", or
- it has double dots "..", or
- it has ASCII control characters, or
- it has ":", "?", "[", "", "^", "~", SP, or TAB anywhere, or
- it has "*" anywhere, or
- it ends with a "/", or
- it ends with ".lock", or
- it contains a "@{" portion

**classmethod create**(*client*, *name*, *force=False*)
Create symlink to object in reference path.

**delete**()
Delete the reference at the given path.

**classmethod iter_items**(*client*, *common_path=None*)
Find all references in the repository.

**name_validator**(*attribute*, *value*)
Validate reference name.

**path**
Return full reference path.

**reference**
Return the path we point to relative to the client.

**rename**(*new_name*, *force=False*)
Rename self to a new name.

**set_reference**(*reference*)
Set ourselves to the given reference path.

## 4.4 Low-level API

This API is built on top of Git and Git-LFS.

Renku repository management.

**class** renku.core.management.**LocalClient**(*path=<function default_path>*, *renku_home='.renku'*, *parent=None*, *use_external_storage=True*, *datadir='data'*, *config_dir='/home/docs/.renku'*, *config_name='renku.ini'*, *lock=None*)
A low-level client for communicating with a local Renku repository.

### 4.4.1 Datasets

Client for handling datasets.

**class** renku.core.management.datasets.**DatasetsApiMixin**(*datadir='data'*)
Client for handling datasets.

**DATASETS = 'datasets'**
Directory for storing dataset metadata in Renku.

**add_data_to_dataset**(*dataset*, *urls*, *git=False*, *force=False*, *sources=()*, *destination=''*, *link=False*)
Import the data into the data directory.

**add_dataset_tag**(*dataset*, *tag*, *description=''*, *force=False*)
Adds a new tag to a dataset.

Validates if the tag already exists and that the tag follows the same rules as docker tags. See https://docs.docker.com/engine/reference/commandline/tag/ for a documentation of docker tag syntax.

**Raises** ValueError

**datadir = None**
Define a name of the folder for storing datasets.

**dataset_commits**(*dataset*, *max_results=None*)
Gets the newest commit for a dataset or its files.

Commits are returned sorted from newest to oldest.

**dataset_path**(*name*)
Get dataset path from name.

**datasets**
Return mapping from path to dataset.

**datasets_from_commit**(*commit=None*)
Return datasets defined in a commit.

**get_dataset**(*path*, *commit=None*)
Return a dataset from a given path.

**get_relative_url**(*url*)
Determine if the repo url should be relative.

**load_dataset**(*name=None*)
Load dataset reference file.

**remove_dataset_tags**(*dataset*, *tags*)
Removes tags from a dataset.

**renku_datasets_path**
Return a `Path` instance of Renku dataset metadata folder.

**with_dataset**(*name=None*, *identifier=None*)
Yield an editable metadata object for a dataset.

renku.core.management.datasets.**check_for_git_repo**(*url*)
Check if a url points to a git repository.

### 4.4.2 Repository

Client for handling a local repository.

**class** renku.core.management.repository.**PathMixin**(*path=<function default_path>*)
Define a default path attribute.

**class** renku.core.management.repository.**RepositoryApiMixin**(*renku_home='.renku'*, *parent=None*)
Client for handling a local repository.

**LOCK_SUFFIX = '.lock'**
Default suffix for Renku lock file.

**METADATA = 'metadata.yml'**
Default name of Renku config file.

**WORKFLOW = 'workflow'**
Directory for storing workflow in Renku.

**cwl_prefix**
Return a CWL prefix.

**find_previous_commit**(*paths*, *revision='HEAD'*, *return_first=False*)
Return a previous commit for a given path starting from `revision`.

**Parameters**

- **revision** – revision to start from, defaults to `HEAD`
- **return_first** – show the first commit in the history

**Raises** **KeyError** – if path is not present in the given commit

**init_repository**(*name=None*, *force=False*)
Initialize a local Renku repository.

**is_cwl**(*path*)
Check if the path is a valid CWL file.

**lock**
Create a Renku config lock.

**parent = None**
Store a pointer to the parent repository.

**process_commit**(*commit=None*, *path=None*)
Build an *Activity*.

**Parameters**

- **commit** – Commit to process. (default: `HEAD`)
- **path** – Process a specific CWL file.

**project**
Return the FOAF/PROV representation of the project.

**project_id**
Return the id for the project based on the repo origin remote.

**remote**
Return host, owner and name of the remote if it exists.

**renku_home = None**
Define a name of the Renku folder (default: `.renku`).

**renku_metadata_path**
Return a `Path` instance of Renku metadata file.

**renku_path = None**
Store a `Path` instance of the Renku folder.

**resolve_in_submodules**(*commit*, *path*)
Resolve filename in submodules.

**subclients**(*parent_commit*)
Return mapping from submodule to client.

**submodules**
Return list of submodules it belongs to.

**with_commit**(*commit*)
Yield the state of the repo at a specific commit.

**with_metadata**(*read_only=False*)
Yield an editable metadata object.

**with_workflow_storage**()
Yield a workflow storage.

**workflow_names**
Return index of workflow names.

**workflow_path**
Return a `Path` instance of the workflow folder.

renku.core.management.repository.**default_path**()
Return default repository path.

### 4.4.3 Git Internals

Wrap Git client.

**class** renku.core.management.git.**GitCore**
Wrap Git client.

**candidate_paths**
Return all paths in the index and untracked files.

**commit**(*author_date=None*, *commit_only=None*, *allow_empty=True*)
Automatic commit.

**dirty_paths**
Get paths of dirty files in the repository.

**ensure_clean**(*ignore_std_streams=False*)
Make sure the repository is clean.

**ensure_unstaged**(*path*)
Ensure that path is not part of git staged files.

**ensure_untracked**(*path*)
Ensure that path is not part of git untracked files.

**find_attr**(**paths*)
Return map with path and its attributes.

**find_ignored_paths**(**paths*)
Return ignored paths matching `.gitignore` file.

**modified_paths**
Return paths of modified files.

**remove_unmodified**(*paths*, *autocommit=True*)
Remove unmodified paths and return their names.

**repo = None**
Store an instance of the Git repository.

**transaction**(*clean=True*, *up_to_date=False*, *commit=True*, *commit_only=None*, *ignore_std_streams=False*, *allow_empty=True*)
Perform Git checks and operations.

**worktree**(*path=None*, *branch_name=None*, *commit=None*, *merge_args=('–ff-only', )*)
Create new worktree.

Git utilities.

**class** renku.core.models.git.**GitURL**(*href*, *pathname=None*, *protocol='ssh'*, *hostname='localhost'*, *username=None*, *password=None*, *port=None*, *owner=None*, *name=None*, *regex=None*)
Parser for common Git URLs.

**image**
Return image name.

**classmethod parse**(*href*)
Derive basic informations.

**class** renku.core.models.git.**Range**(*start*, *stop*)
Represent parsed Git revision as an interval.

**classmethod rev_parse**(*git*, *revision*)
Parse revision string.

renku.core.models.git.**filter_repo_name**(*repo_name*)
Remove the .git extension from the repo name.

# Python Module Index

## r

# Index

## D

## E



## F

## G



## I



## L



## M

## N



## O



## P



## R

## S



## T

## U



## V



## W