renku Documentation

Release 0.4.0

Swiss Data Science Center

May 23, 2019

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 --dev 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 MacOS

The recommended way of installing Renku on MacOS 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.

For more information about the Renku API see its documentation.

1.4 Use the Renku command line

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 command line* instructions.

1.4.1 Renku Command Line

The base command for interacting with the Renku platform.

`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.

`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
```

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
```

renku datasets

Work with datasets in the current repository.

Manipulating datasets

Creating an empty dataset inside a Renku project:

```
$ renku dataset create my-dataset
```

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 a specific path within the parent project. In this case, use the `--target` flag:

```
$ renku dataset add my-dataset --target relative-path/datafile \
    git+ssh://host.io/namespace/project.git
```

To trim part of the path from the parent directory, use the `--relative-to` option. For example, the command above will result in a structure like

```
data/
  my-dataset/
    relative-path/
      datafile
```

Using instead

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

will yield:

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

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`.

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 understading 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.

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.

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

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

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
```

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 `@`.

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
```

`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*.

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.

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.

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
```

(continues on next page)

(continued from previous page)

```
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
```

renku storage

Manage an external storage.

renku image

Manipulate images related to the Renku project.

Configure the image registry

First, obtain an access token for the registry from GitLab by going to `<gitlab-URL>/profile/personal_access_tokens`. Select only the `read_registry` scope and copy the access token.

```
$ open https://<gitlab-URL>/profile/personal_access_tokens
$ export ACCESS_TOKEN=<copy-from-browser>
```

Find your project's registry path by going to `<gitlab-url>/<namespace>/<project>/container_registry`. The string following the docker push command is the `registry-path` for the project.

```
$ open https://<gitlab-url>/<namespace>/<project>/container_registry
$ renku config registry https://oauth2:$ACCESS_TOKEN@<registry-path>
```

You can use any registry with manual authentication step using Docker command line.

```
$ docker login docker.io
$ renku config registry https://docker.io
```

Pull image

If the image has indeed been built and pushed to the registry, you should be able to fetch it with:

```
$ renku image pull
```

This pulls an image that was built for the current commit. You can also fetch an image built for a specific commit with:

```
# renku image pull --revision <ref-name>
$ renku image pull --revision HEAD~1
```

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".
```

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.

1.4.2 Projects

Model objects representing projects.

class renku.models.projects.**Project**(*name=None*, *created=NOTHING*, *updated=NOTHING*, *version='1'*)

Represent a project.

Type:

```
"foaf:Project"
```

Context:

```
{
  "foaf": "http://xmlns.com/foaf/0.1/",
  "name": "foaf:name",
  "created": "http://schema.org/dateCreated",
  "updated": "http://schema.org/dateUpdated",
  "version": "http://schema.org/schemaVersion"
}
```

class renku.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.models.projects.Project*

1.4.3 Datasets

Manage datasets and their metadata.

Dataset object

class renku.models.datasets.**Dataset**(*name: str*, *created=NOTHING*, *identifier=NOTHING*, *authors=NOTHING*, *files=NOTHING*)

Repesent a dataset.

Type:

```
"dctypes:Dataset"
```

Context:

```
{
  "dcterms": "http://purl.org/dc/terms/",
  "dctypes": "http://purl.org/dc/dcmitypes/",
  "foaf": "http://xmlns.com/foaf/0.1/",
  "prov": "http://www.w3.org/ns/prov#",
  "scoro": "http://purl.org/spar/scoro/",
  "name": "dcterms:name",
  "created": "http://schema.org/dateCreated",
  "identifier": {
    "@id": "dctypes:Dataset",
    "@type": "@id"
  },
  "authors": {
    "@container": "@list"
  },
  "email": "dcterms:email",
  "affiliation": "scoro:affiliate",
  "files": {
    "@container": "@index"
  },
  "url": "http://schema.org/url",
  "added": "http://schema.org/dateCreated"
}
```

authors_csv
Comma-separated list of authors associated with dataset.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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

short_id
Shorter version of identifier.

Dataset file

Manage files in the dataset.

class renku.models.datasets.**DatasetFile**(*path*, *url=None*, *authors=NOTHING*, *dataset=None*, *added=NOTHING*)

Represent a file in a dataset.

Type:

```
"http://schema.org/DigitalDocument"
```

Context:

```
{
  "url": "http://schema.org/url",
  "authors": {
    "@container": "@list"
  },
  "foaf": "http://xmlns.com/foaf/0.1/",
  "dcterms": "http://purl.org/dc/terms/",
  "scoro": "http://purl.org/spar/scoro/",
  "name": "dcterms:name",
  "email": "dcterms:email",
  "affiliation": "scoro:affiliate",
  "added": "http://schema.org/dateCreated"
}
```

classmethod from_jsonld(*data*)

Instantiate a JSON-LD class from data.

Author

class renku.models.datasets.**Author**(*name*, *email*, *affiliation=None*)

Represent the author of a resource.

Type:

```
"dcterms:creator"
```

Context:

```
{
  "foaf": "http://xmlns.com/foaf/0.1/",
  "dcterms": "http://purl.org/dc/terms/",
  "scoro": "http://purl.org/spar/scoro/",
  "name": "dcterms:name",
  "email": "dcterms:email",
  "affiliation": "scoro:affiliate"
}
```

check_email(*attribute*, *value*)

Check that the email is valid.

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*)

Instantiate a JSON-LD class from data.

1.4.4 Provenance

Extract provenance information from the repository.

Activities

class renku.models.provenance.activities.**Activity**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=NOTHING*, *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*)

Represent an activity in the repository.

Type:

```
"prov:Activity"
```

Context:

```
{
  "dcterms": "http://purl.org/dc/terms/",
  "prov": "http://www.w3.org/ns/prov#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "dcterms: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"
  }
}
```

default_ended_at_time()
Configure calculated properties.

default_generated()
Calculate default values.

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_project()
Generate a default location.

default_started_at_time()
Configure calculated properties.

default_was_informed_by()
List parent actions.

static from_git_commit(*commit*, *client*, *path=None*)
Populate information from the given Git commit.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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.

class renku.models.provenance.activities.**ProcessRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=NOTHING*, *id=NOTHING*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *process=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=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#",
  "dcterms": "http://purl.org/dc/terms/",
  "prov": "http://www.w3.org/ns/prov#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "dcterms: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"
  },
  "association": "prov:qualifiedAssociation",
  "qualified_usage": "prov:qualifiedUsage"
}
```

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_project()
Generate a default location.

default_qualified_usage()
Generate list of used artifacts.

default_started_at_time()
Configure calculated properties.

default_was_informed_by()
List parent actions.

static from_git_commit(*commit*, *client*, *path=None*)
Populate information from the given Git commit.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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.

class renku.models.provenance.activities.**WorkflowRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project=NOTHING*, *id=NOTHING*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *process=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=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#",
  "dcterms": "http://purl.org/dc/terms/",
  "prov": "http://www.w3.org/ns/prov#",
```

(continues on next page)

(continued from previous page)

```
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "path": "prov:atLocation",
  "_label": "rdfs:label",
  "_project": "dcterms: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"
  },
  "association": "prov:qualifiedAssociation",
  "qualified_usage": "prov:qualifiedUsage",
  "_processes": {
    "@reverse": "wfprov:wasPartOfWorkflowRun"
  }
}
```

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_project()
Generate a default location.

default_qualified_usage()
Generate list of used artifacts.

default_started_at_time()
Configure calculated properties.

default_subprocesses()
Load subprocesses.

default_was_informed_by()
List parent actions.

static from_git_commit(*commit*, *client*, *path=None*)
Populate information from the given Git commit.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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.

Entities and Plans

class renku.models.provenance.entities.**Entity**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=NOTHING*, *parent=None*)
Represent a data value or item.

Type:

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

Context:

```
{
  "dcterms": "http://purl.org/dc/terms/",
  "prov": "http://www.w3.org/ns/prov#",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "dcterms:isPartOf"
}
```

default_id()
Configure calculated ID.

default_label()
Generate a default label.

default_project()
Generate a default location.

entities
Yield itself.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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

parent
Return the parent object.

submodules
Proxy to client submodules.

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

Represent a directory with files.

Type:

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

Context:

```
{
  "prov": "http://www.w3.org/ns/prov#",
  "dcterms": "http://purl.org/dc/terms/",
  "wfprov": "http://purl.org/wf4ever/wfprov#",
  "path": "prov:atLocation",
  "_id": "@id",
  "_label": "rdfs:label",
  "_project": "dcterms:isPartOf",
  "members": "prov:hadMember"
}
```

default_id()
Configure calculated ID.

default_label()
Generate a default label.

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

default_project()
Generate a default location.

entities
Recursively return all files.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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

parent
Return the parent object.

submodules
Proxy to client submodules.

class renku.models.provenance.entities.**Process**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=NOTHING*, *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": "dcterms:isPartOf",
  "_activity": "prov:activity"
}
```

activity
Return the activity object.

default_id()
Configure calculated ID.

default_label()
Generate a default label.

default_project()
Generate a default location.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

submodules
Proxy to client submodules.

class renku.models.provenance.entities.**Workflow**(*, *commit=None*, *client=None*, *path=None*, *id=NOTHING*, *label=NOTHING*, *project=NOTHING*, *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": "dcterms:isPartOf",
  "_activity": "prov:activity",
  "subprocesses": "wfdesc:hasSubProcess"
}
```


activity
Return the activity object.

default_id()
Configure calculated ID.

default_label()
Generate a default label.

default_project()
Generate a default location.

default_subprocesses()
Load subprocesses.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

submodules
Proxy to client submodules.

Agents

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

Type:


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


Context:


```
{
  "foaf": "http://xmlns.com/foaf/0.1/",
  "prov": "http://purl.org/dc/terms/",
  "name": "rdfs:label",
  "email": {
    "@type": "@id",
    "@id": "foaf:mbox"
  },
  "_id": "@id"
}
```


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

default_id()
Configure calculated ID.

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

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

class renku.models.provenance.agents.**SoftwareAgent**(***, *label*, *was_started_by=None*, *id*)
Represent a person.

Type:

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

Context:

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

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

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

Relations

class renku.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"
}
```

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

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

class renku.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.

default_id()
Configure calculated ID.

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

Expanded

class renku.models.provenance.expanded.**Project**(*, *id*)
Represent a project.

Type:

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

Context:

```
{
  "foaf": "http://xmlns.com/foaf/0.1/",
  "prov": "http://www.w3.org/ns/prov#",
  "_id": "@id"
}
```

classmethod from_jsonld(*data*)
Instantiate a JSON-LD class from data.

1.4.5 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.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.

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.models.cwl.command_line_tool.**CommandLineToolFactory**(*command_line*, *directory='.'*, *working_dir='.'*, *stdin=None*, *stderr=None*, *stdout=None*, *successCodes=NOTHING*)

Command Line Tool Factory.

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

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.

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*, *outputs=None*)
Watch a Renku repository for changes to detect outputs.

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

Parameter

Represent parameters from the Common Workflow Language.

class renku.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()
Format command input parameter as shell argument.

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

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

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

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

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

Convert a default value.

Process

Represent a `Process` from the Common Workflow Language.

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

Represent a directory.

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

Define a file or subdirectory.

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

Represent a file.

Workflow

Represent workflows from the Common Workflow Language.

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

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

1.4.6 Low-level API

This API is built on top of REST API endpoints exposed by Renku services.

Warning: Renku services are currently in **beta preview** status and they are subject to change in forseenable future.

HTTP clients for Renku platform.

class renku.api.**LocalClient**(*path=<function default_path>*, *renku_home='.renku'*, *parent=None*, *use_external_storage=True*, *datadir='data'*)
A low-level client for communicating with a local Renku repository.

Example:

```
>>> import renku
>>> client = renku.LocalClient('.')
```

Datasets

Client for handling datasets.

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

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

add_data_to_dataset(*dataset*, *url*, *git=False*, *force=False*, ***kwargs*)
Import the data into the data directory.

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

datasets
Return mapping from path to dataset.

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

renku_datasets_path
Return a Path instance of Renku dataset metadata folder.

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

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

Repository

Client for handling a local repository.

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

class renku.api.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 a previous commit for a given path.

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.

project
Return FOAF/PROV representation of the project.

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_metadata()
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.api.repository.**default_path**()
Return default repository path.

1.4.7 Contributing

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

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 :)

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.

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.

1.4.8 Changes

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.

v0.3.3

(released 2018-12-07)

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

v0.3.2

(released 2018-11-29)

- Fixes display of workflows in `renku log`.

v0.3.1

(released 2018-11-29)

- Fixes issues with parsing remote Git URLs.

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.

v0.2.0

(released 2018-09-25)

- Refactored version using Git and Common Workflow Language.

v0.1.0

(released 2017-09-06)

- Initial public release as Renga.

1.4.9 License

```
Copyright 2017-2018 - Swiss Data Science Center (SDSC)
A partnership between École Polytechnique Fédérale de Lausanne (EPFL) and
Eidgenössische Technische Hochschule Zürich (ETHZ).

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

1.4.10 Authors

Python SDK and CLI for the Renku platform.

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

Python Module Index

r

Index

E

F

G

S

T

U

V

W