renku Documentation

Release 0.11.1

Swiss Data Science Center

Sep 08, 2020

Contents

A Python library for the Renku collaborative data science platform. It includes a CLI and SDK for end-users as well as a service backend. It provides functionality for the creation and management of projects and datasets, and simple utilities to capture data provenance while performing analysis tasks.

NOTE: `renku-python` is the python library and core service for Renku - it *does not* start the Renku platform itself - for that, refer to the Renku docs on running the platform.

CHAPTER 1

Renku for Users

1.1 Installation

Renku releases and development versions are available from PyPI. You can install it using any tool that knows how to handle PyPI packages. Our recommendation is to use :code:pipx.

1.1.1 `pipx`

First, 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
```

`pipx` installs renku into its own virtual environment, making sure that it does not pollute any other packages or versions that you may have already installed.

Note: If you install renku as a dependency in a virtual environment and the environment is active, your shell will default to the version installed in the virtual environment, *not* the version installed by `pipx`.

To install a development release:

```
$ pipx install --pip-args pre renku
```

1.1.2 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.

1.1.3 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 issues / contacting developers

Report bugs on our issue tracker.

If you want to submit a bug, improvement or feature suggestions feel free to open a corresponding issue on GitHub.

If you are reporting a bug, please help us to speed up the diagnosing a problem by providing us with as much as information as possible. Ideally, that would include a step by step process on how to reproduce the bug.

If you have a general usage question please ask it on our discourse forum or our chat.

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 proposal discussions or epics and feel free to express your proposal on the topic. Once topic has been flushed out and we have decided how feature should be implemented, we can start implementing them.

Improvement requests

If you see room for improvement, please open an issue with a suggestion. Please motivate your suggestion by illustrating a problem it solves.

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. Ensure you have your development environment set up. For this we encourage usage of *pipenv* and *pyenv*:

```
$ pyenv install 3.7.5rc1
$ cd renku/
$ pipenv install --python  ~/.pyenv/versions/3.7.5rc1/bin/python
$ pipenv shell
```

4. Create a branch for local development:

```
$ git checkout -b <issue_number>_<short_description>
```

   Now you can make your changes locally.

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

```
$ pipenv run tests
```

   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 black.

```
$ black .
```

   You may want to set up black styling as a pre-commit hook to do this automatically. Install pre-commit and run the following command in the root of the *renku-python* repository:

```
$ pre-commit install
```

   See https://github.com/psf/black#version-control-integration for more information. Make sure to remove other formatting hooks (e.g. yapf) if you already have them set up.

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

```
$ git add .
$ git commit -s
    -m "type(scope): 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 Commit message guidelines

This project is using conventional commits style for generation of changelog upon each release. Therefore, it's important that our commit messages convey what they do correctly. Commit message should always follow this pattern:

    $ %{type}(%{scope}): %{description}

Type's used for describing commit's which will end up in changelog are `fix:` & `feat:`.

Please note that the `fix` type here is only for user-facing bug fixes and not fixes on tests or CI. For those, please use: `ci:` or `test:`

Full list of types which are in use:

- `feat:` - Used for new user-facing features. This should be related to one of the predefined scopes. If a scope does not exist, a new scope may be proposed.
- `fix:` - Used for fixing user-facing bugs. This should be related to one of the predefined scopes.
- `chore:` - Used for changes which are not user-facing. The scope should be a module name in which chore occurred.
- `tests:` - Used for fixing existing or adding new tests. The scope should relate to a predefined scope or be omitted.
- `docs:` - Used for adding more documentation. If documentation is not related to predefined user scopes, it can be omitted.
- `refactor` - Used for changing the code structure. Scope being used here should be module name. If refactoring is across multiple modules, scope could be omitted or PR broken down into smaller chunks.

Full list of user-facing scopes which are in use:

- `graph` - Scope for describing knowledge graph which is being build with users usage of the system.
- `workflow` - Scope for describing reproducibility flow.
- `dataset` - Scope for describing datasets.
- `core` - General scope for describing all core elements of Renku project.
- `service` - General scope for describing interaction or operation of Renku as a service.
- `cli` - General scope for describing interaction through CLI.

PLEASE NOTE: Types which are defined to be picked up for change log (`feat:` and `fix:`) should always contain a scope due to grouping which occurs in changelog when we have them.

3.3.4 Pull Request Guidelines

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

- Make sure you agree with the license and follow the legal matter.
- The pull request should include tests and must not decrease test coverage.
- If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
- The pull request should work for Python 3.6, 3.7 and 3.8. Check GitHub action builds and make sure that the tests pass for all supported Python versions.

3.4 Changes

3.4.1 0.11.1 (2020-08-18)

Bug Fixes

- fixes shacl for DatasetFile when used inside a qualifiedGeneration (#1477) (99dd4a4)

3.4.2 0.11.0 (2020-08-14)

Bug Fixes

- **cli:** disable version check in githook calls (#1300) (5132db3)
- **core:** fix paths in migration of workflows (#1371) (8c3d34b)
- **core:** Fixes SoftwareAgent person context (#1323) (a207a7f)
- **core:** Only update project metadata if any migrations were executed (#1308) (1056a03)
- **service:** adds more custom logging and imp. except handling (#1435) (6c3adb5)
- **service:** fixes handlers for internal loggers (#1433) (a312f7c)
- **service:** move project_id to query string on migrations check (#1367) (0f89726)
- **tests:** integration tests (#1351) (3974a39)

Features

- **cli:** Adds renku save command (#1273) (4ddc1c2)
- **cli:** prompt for missing variables (1e1d408), closes #1126
- **cli:** Show detailed commands for renku log output (#1345) (19fb819)
- **core:** Calamus integration (#1281) (bda538f)
- **core:** configurable data dir (#1347) (e388773)
- **core:** disabling of inputs/outputs auto-detection (#1406) (3245ca0)
- **core:** migration check in core (#1320) (4bc52f4)
- **core:** Move workflow serialisation over to calamus (#1386) (f0fbc49)
- **core:** save and load workflow as jsonld (#1185) (d403289)
- **core:** separate models for migrations (#1431) (127d606)
- **dataset:** source and url for DatasetFile (#1451) (b4fa5db)
- **service:** added endpoints to execute all migrations on a project (#1322) (aca8cc2)
- **service:** adds endpoint for explicit migrations check (#1326) (146b1a7)
- **service:** adds source and destination versions to migrations check (#1372) (ea76b48)
- decode base64 headers (#1407) (9901cc3)
- **service:** adds endpoints for dataset remove (#1383) (289e4b9)
- **service:** adds endpoints for unlinking files from a dataset (#1314) (1b78b16)
- **service:** async migrations execution (#1344) (ff66953)
- **service:** create new projects from templates (#1287) (552f85c), closes #862

3.4.3 0.10.5 (2020-07-16)

Bug Fixes

- **core:** Pin dependencies to prevent downstream dependency updates from breaking renku. Fix pyshacl dependency. (#785) (30beedd)
- **core:** Fixes SoftwareAgent person context. (#1323) (fa62f58)

3.4.4 0.10.4 (2020-05-18)

Bug Fixes

- **dataset:** update default behaviour and messaging on dataset unlink (#1275) (98d6728)
- **dataset:** correct url in different domain (#1211) (49e8b8b)

Features

- **cli:** Adds warning messages for LFS, fix output redirection (#1199) (31969f5)
- **core:** Adds lfs file size limit and lfs ignore file (#1210) (1f3c81c)
- **core:** Adds renku storage clean command (#1235) (7029400)
- **core:** git hook to avoid committing large files (#1238) (e8f1a8b)
- **core:** renku doctor check for lfs migrate info (#1234) (480da06)
- **dataset:** fail early when external storage not installed (#1239) (e6ea6da)
- **core:** project clone API support for revision checkout (#1208) (74116e9)
- **service:** protected branches support (#1222) (8405ce5)
- **dataset:** doi variations for import (#1216) (0f329dd)
- **dataset:** keywords in metadata (#1209) (f98a800)
- **dataset:** no failure when adding ignored files (#1213) (b1e275f)
- **service:** read template manifest (#1254) (7eac85b)

3.4.5 0.10.3 (2020-04-22)

Bug Fixes

- **dataset:** avoid recursive addition of data directory (#1163) (79e3b03)
- **dataset:** commit after unlinking files (#1120) (97e8754)
- **dataset:** Dataverse export (#1028) (737cecf)

Features

- **core:** CLI warning when in non-root directory (#1162) (115e462)
- **dataset:** migrate submodule-based datasets (#1092) (dba20c4)
- **dataset:** no failure when adding existing files (#1177) (a68dcb7)
- **dataset:** remove –link flag (#1164) (969d4f8)
- **dataset:** show file size in ls-files (#1123) (0951930)
- **datasets:** specify title on dataset creation (#1204) (fb70ac5)
- **init:** read and display template variables (#1134) (0f86dc5), closes #1126
- **service:** add remote files to dataset (#1139) (f6bebfe)

3.4.6 0.10.1 (2020-03-31)

Bug Fixes

- **core:** directory input regression (#1155) (b17c843)
- **dataset:** correct url when importing with DOI (#1156) (025b735)

Features

- renku init options refactor (#1127) (78b208b)
- **datasets:** add files from dropbox (#1135) (bf5f2db)

3.4.7 0.10.0 (2020-03-25)

This release brings about several important Dataset features:

- importing renku datasets (#838)
- working with data external to the repository (#974)
- editing dataset metadata (#1111)

Please see the Dataset documentation for details.

Additional features were implemented for the backend service to facilitate a smoother user experience for dataset file manipulation.

IMPORTANT: starting with this version, a new metadata migration mechanism is in place (#1003). Renku commands will insist on migrating a project immediately if the metadata is found to be outdated.

Bug Fixes

- **cli:** consistenly show correct contexts (#1096) (b333f0f)
- **dataset:** –no-external-storage flag not working (#1130) (c183e97)
- **dataset:** commit only updated dataset files (#1116) (d9739df)
- **datasets:** fixed importing large amount of small files (#1119) (8d61473)

- **datasets:** raises correct error message on import of protected dataset (#1112) (e579904)

Features

- **core:** new migration mechanism (#1003) (1cc33d4)
- **dataset:** adding external data without copying (#974) (6a17512)
- **dataset:** bypass import confirmation (#1124) (947210a)
- **dataset:** import renku datasets (#838) (6aa3651)
- **dataset:** metadata edit (#1111) (66cfbbc)
- **dataset:** wildcard support when adding data from git (#1128) (baa1c9f)

3.4.8 0.9.1 (2020-02-24)

Bug Fixes

- added test utility functions and cleanup (#1014) (f41100d)
- cache instance cleanup (#1051) (12f5446)
- enable dataset cmd in sub directories (#1012) (e3191e1)
- fields with default need to come last (#1046) (649b159)
- fixes renku show sibling handling with no paths (#1026) (8df678f)
- flush old keys for user projects and files (#1002) (7438c73)
- generate https IDs for entities instead of file:// (#1009) (87f7750)
- handle errors correctly (#1040) (950eeac)
- improved list datasets and files (#1034) (fd96d68)
- pin idna to 2.8 (#1020) (19ea7af)
- resync repo after import action (#1052) (b38341b)
- standardize test assertions (#1016) (16e8e63)
- temporarily disable integration tests (#1036) (8c8fd7a)
- updated readme to include local testing (#1000) (351a650)
- run tests via pipenv run commands (#999) (d8095e3)

Features

- **svc:** adds job details endpoint (#1050) (9c58a08)
- **svc:** added list user jobs endpoint (#1001) (f3c200c)
- **svc:** dataset import via service (#1023) (d6c670a)

3.4.9 0.9.0 (2020-02-07)

Bug Fixes

- adds git user check before running renku init (#892) (2e52dff)
- adds sorting to file listing (#960) (bcf6bcd)
- avoid empty commits when adding files (#842) (8533a7a)
- Fixes dataset naming (#898) (418deb3)
- Deletes temporary branch after renku init –force (#887) (eac0463)
- enforces label on SoftwareAgent (#869) (71badda)
- Fixes JSON-LD translation and related issues (#846) (65e5469)
- Fixes renku run error message handling (#961) (81d31ff)
- Fixes renku update workflow failure handling and renku status error handling (#888) (3879124)
- Fixes sameAs property to follow schema.org spec (#944) (291380e)
- handle missing renku directory (#989) (f938be9)
- resolves symlinks when pulling LFS (#981) (68bd8f5)
- serializes all zenodo metadata (#941) (787978a)
- Fixes various bugs in dataset import (#882) (be28bf5)

Features

- add project initialization from template (#809) (4405744)
- added renku service with cache and datasets (#788) (7a7068d), closes #767 #846
- Adds protection for renku relevant paths in dataset add (#939) (a3c02e8)
- Adds prov:Invalidated output to renku log (008ab20)
- better UX when adding to a dataset (#911) (c6ac967)
- check for required git hooks (#854) (54ba91d)
- Dataverse export (#909) (7e9e647)
- improve dataset command output (#927) (c7639d3)
- metadata on dataset creation (#850) (b357ee7)
- Plugin support for renku-run (#883) (7dbda83)
- python 3.8 compatibility (#861) (4aaac8d)
- SHACL Validation (#767) (255a01d)
- update bug_report template to be more renku-relevant (#988) (e00ded7)

3.4.10 0.8.0 (2019-11-21)

Bug Fixes

- addressed CI problems with git submodules (#783) (0d3eeb7)
- adds simple check on empty filename (#786) (8cd061b)
- ensure all Person instances have valid ids (4f80efc), closes #812
- Fixes jsonld issue when importing from dataverse (#759) (ffe36c6)
- fixes nested type scoped handling if a class only has a single class (#804) (16d03b6)
- ignore deleted paths in generated entities (86fedaf), closes #806
- integration tests (#831) (a4ad7f9)
- make Creator a subclass of Person (ac9bac3), closes #793
- Redesign scoped context in jsonld (#750) (2b1948d)

Features

- avoid creation of nested datasets (#796) (6084c87)
- do not create dataset implicitly (#779) (84e59d0)
- local git repo not treated as remote (8cc2834)
- renku clone command (#828) (4b3b615)

3.4.11 0.7.0 (2019-10-15)

Bug Fixes

- use UI-resolved project path as project ID (#701) (dfcc9e6)

3.4.12 0.6.1 (2019-10-10)

Bug Fixes

- add .renku/tmp to default .gitignore (#728) (6212148)
- dataset import causes renku exception due to duplicate LocalClient (#724) (89411b0)
- delete new dataset ref if file add fails (#729) (2dea711)
- fixes bug with deleted files not getting committed (#741) (5de4b6f)
- force current project for entities (#707) (538ef07)
- integration tests for #681 (#747) (b08435d)
- use commit author for project creator (#715) (1a40ebe), closes #713
- zenodo dataset import error (f1d623a)

Features

- adds basic QA checklist (#698) (c97e9bd)
- dataset tagging (#693) (797161f)
- include creator in project metadata (#687) ([9c7753e](https://git

3.4.13 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)

3.4.14 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

3.4.15 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)

3.4.16 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)

3.4.17 `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.

3.4.18 v0.3.3

(released 2018-12-07)

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

3.4.19 v0.3.2

(released 2018-11-29)

- Fixes display of workflows in `renku log`.

3.4.20 v0.3.1

(released 2018-11-29)

- Fixes issues with parsing remote Git URLs.

3.4.21 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.

3.4.22 `v0.2.0`

(released 2018-09-25)

- Refactored version using Git and Common Workflow Language.

3.4.23 `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.
  --global-config-path            Print global application's config path.
  --install-completion            Install completion for the current shell.
  --path <path>                   Location of a Renku repository.
                                  [default: (dynamic)]
  --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 --global-config-path`.

4.2.2 `renku init`

Create an empty Renku project or reinitialize an existing one.

Start 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.

Use a different template

Renku is installed together with a specific set of templates you can select when you initialize a project. You can check them by typing:

```
$ renku init --list-templates

INDEX ID     DESCRIPTION                    PARAMETERS
----- ------ ------------------------------ ------------------------------
1     python The simplest Python-based [...] description: project des[...]
2     R      R-based renku project with[...] description: project des[...]
```

If you know which template you are going to use, you can provide either the id `--template-id` or the template index number `--template-index`.

You can use a newer version of the templates or even create your own one and provide it to the `init` command by specifying the target template repository source `--template-source` (both local path and remote url are supported) and the reference `--template-ref` (branch, tag or commit).

You can take inspiration from the official Renku template repository

```
$ renku init --template-ref master --template-source \
https://github.com/SwissDataScienceCenter/renku-project-template

Fetching template from
https://github.com/SwissDataScienceCenter/renku-project-template@master
... OK

INDEX ID              DESCRIPTION                PARAMETERS
----- --------------- -------------------------- ----------------------
1     python-minimal  Basic Python Project:[...] description: proj[...]
2     R-minimal       Basic R Project: The [...] description: proj[...]
```

(continues on next page)

(continued from previous page)

```
Please choose a template by typing the index:
```

Provide parameters ~~~~~~~~~~~~~~~~~~-

Some templates require parameters to properly initialize a new project. You can check them by listing the templates `--list-templates`.

To provide parameters, use the `--parameter` option and provide each parameter using `--parameter "param1"="value1"`.

```
$ renku init --template-id python-minimal --parameter \
"description"="my new shiny project"

Initializing new Renku repository... OK
```

If you don't provide the required parameters through the option `-parameter`, you will be asked to provide them. Empty values are allowed and passed to the template initialization function.

Note: Every project requires a `name` that can either be provided using `--name` or automatically taken from the target folder. This is also considered as a special parameter, therefore it's automatically added to the list of parameters forwarded to the `init` command.

Update 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
```

By default, configuration is stored locally in the project's directory. Use `--global` option to store configuration for all projects in your home directory.

Remove values

To remove a specific key from configuration use:

```
$ renku config --remove registry
```

By default, only local configuration is searched for removal. Use `--global` option to remove a global configuration value.

Query values

You can display all configuration values with:

```
$ renku config
```

Both local and global configuration files are read. Values in local configuration take precedence over global values. Use `--local` or `--global` flag to read corresponding configuration only.

You can provide a KEY to display only its value:

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

Available configuration values

The following values are available for the `renku config` command:

Name	Description	Default
registry	The image registry to store Docker images in	`None`
zenodo.access_token	Access token for Zenodo API	`None`
dataverse.access_token	Access token for Dataverse API	`None`
dataverse.server_url	URL for the Dataverse API server to use	`None`
show_lfs_message	Whether to show messages about files being added to git LFS or not	`True`
lfs_threshold	Threshold file size below which files are not added to git LFS	`100kb`

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
```

You can pass the following options to this command to set various metadata for the dataset.

Option	Description
-t, –title	A human-readable title for the dataset.
-d, –description	Dataset's description.
-c, –creator	Creator's name, email, and an optional affiliation. Accepted format is 'Forename Surname <email> [affiliation]'. Pass multiple times for a list of creators.
-k, –keyword	Dataset's keywords. Pass multiple times for a list of keywords.

Editing a dataset's metadata

Use `edit` subcommand to change metadata of a dataset. You can edit the same set of metadata as the create command by passing the options described in the table above.

```
$ renku dataset edit my-dataset --title 'New title'
Successfully updated: title.
```

Listing all datasets:

```
$ renku dataset
ID        NAME           TITLE          VERSION
--------  -------------  -------------  ---------
0ad1cb9a  some-dataset   Some Dataset
9436e36c  my-dataset     My Dataset
```

You can select which columns to display by using `--columns` to pass a comma-separated list of column names:

```
$ renku dataset --columns id,name,date_created,creators
ID        NAME           CREATED              CREATORS
--------  -------------  -------------------  ---------
0ad1cb9a  some-dataset   2020-03-19 16:39:46  sam
9436e36c  my-dataset     2020-02-28 16:48:09  sam
```

Displayed results are sorted based on the value of the first column.

To inspect the state of the dataset on a given commit we can use `--revision` flag for it:

```
$ renku dataset --revision=1103a42bd3006c94efcaf5d6a5e03a335f071215
ID        NAME                 TITLE               VERSION
a1fd8ce2  201901_us_flights_1  2019-01 US Flights  1
c2d80abe  ds1                  ds1
```

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.

You can create a dataset when you add data to it for the first time by passing `--create` flag to add command:

```
$ renku dataset add --create new-dataset http://data-url
```

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 add 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 shell-like wildcards (e.g. , *, ?) when specifying paths to be added. Put wildcard patterns in quotes to prevent your shell from expanding them.

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

You can use `--destination` or `-d` flag to set the location where the new data is copied to. This location be will under the dataset's data directory and will be created if does not exists. You will get an error message if the destination exists and is a file.

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

will yield:

```
data/
  my-dataset/
    new-dir/
      new-subdir/
        datafile
```

To add a specific version of files, use `--ref` option for selecting a branch, commit, or tag. The value passed to this option must be a valid reference in the remote Git repository.

Adding external data to the dataset:

Sometimes you might want to add data to your dataset without copying the actual files to your repository. This is useful for example when external data is too large to store locally. The external data must exist (i.e. be mounted) on your filesystem. Renku creates a symbolic to your data and you can use this symbolic link in renku commands as a normal file. To add an external file pass `--external` or `-e` when adding local data to a dataset:

```
$ renku dataset add my-dataset -e /path/to/external/file
```

Updating a dataset:

After adding files from a remote Git repository, you can check for updates in those files by using `renku dataset update` command. This command checks all remote files and copies over new content if there is any. It does not delete files from the local dataset if they are deleted from the remote Git repository; to force the delete use `--delete` argument. You can update to a specific branch, commit, or tag by passing `--ref` option.

You can limit the scope of updated files by specifying dataset names, using `--include` and `--exclude` to filter based on file names, or using `--creators` to filter based on creators. For example, the following command updates only CSV files from `my-dataset`:

```
$ renku dataset update -I '*.csv' my-dataset
```

Note that putting glob patterns in quotes is needed to tell Unix shell not to expand them.

External data are not updated automatically because they require a checksum calculation which can take a long time when data is large. To update external files pass `--external` or `-e` to the update command:

```
$ renku dataset update -e
```

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
-------------------  ------  ---------------  ----------  ----------------
2020-09-19 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 other Renku projects:

To import all data files and their metadata from another Renku dataset use:

```
$ renku dataset import \
    https://renkulab.io/projects/<username>/<project>/datasets/<dataset-id>
```

or

```
$ renku dataset import \
    https://renkulab.io/datasets/<dataset-id>
```

You can get the link to a dataset form the UI or you can construct it by knowing the dataset's ID.

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.

To export to a Dataverse provider you must pass Dataverse server's URL and the name of the parent dataverse where the dataset will be exported to. Server's URL is stored in your Renku setting and you don't need to pass it every time.

Listing all files in the project associated with a dataset.

```
$ renku dataset ls-files
DATASET NAME         ADDED                PATH
-------------------  -------------------  ----------------------------
my-dataset           2020-02-28 16:48:09  data/my-dataset/addme
my-dataset           2020-02-28 16:49:02  data/my-dataset/weather/file1
my-dataset           2020-02-28 16:49:02  data/my-dataset/weather/file2
my-dataset           2020-02-28 16:49:02  data/my-dataset/weather/file3
```

You can select which columns to display by using `--columns` to pass a comma-separated list of column names:

```
$ renku dataset ls-files --columns name,creators, path
DATASET NAME         CREATORS   PATH
-------------------  ---------  ----------------------------
my-dataset           sam        data/my-dataset/addme
my-dataset           sam        data/my-dataset/weather/file1
my-dataset           sam        data/my-dataset/weather/file2
my-dataset           sam        data/my-dataset/weather/file3
```

Displayed results are sorted based on the value of the first column.

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

```
$ renku dataset ls-files --include "file*" --exclude "file3"
DATASET NAME         ADDED                PATH
-------------------  -------------------  ----------------------------
my-dataset           2020-02-28 16:49:02  data/my-dataset/weather/file1
my-dataset           2020-02-28 16:49:02  data/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`.

Warning: Circular dependencies are not supported for `renku run`. See *Circular Dependencies* for more details.

Warning: When using output redirection in `renku run` on Windows (with ‘‘ > file‘‘ or ‘‘ 2> file‘‘), all Renku errors and messages are redirected as well and `renku run` produces no output on the terminal. On Linux, this is detected by renku and only the output of the command to be run is actually redirected. Renku specific messages such as errors get printed to the terminal as usual and don’t get redirected.

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. Explicit inputs 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.

Disabling input detection (`--no-input-detection`)

Input paths detection can be disabled by passing `--no-input-detection` flag to `renku run`. In this case, only the directories/files that are passed as explicit input are considered to be file inputs. Those passed via command arguments are ignored unless they are in the explicit inputs list. This only affects files and directories; command options and flags are still treated as inputs.

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.

Disabling output detection (`--no-output-detection`)

Output paths detection can be disabled by passing `--no-output-detection` flag to `renku run`. When disabled, only the directories/files that are passed as explicit output are considered to be outputs and those passed via command arguments are ignored.

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
```

Circular Dependencies

Circular dependencies are not supported in `renku run`. This means you cannot use the same file or directory as both an input and an output in the same step, for instance reading from a file as input and then appending to it is not allowed. Since renku records all steps of an analysis workflow in a dependency graph and it allows you to update outputs when an input changes, this would lead to problems with circular dependencies. An update command would change the input again, leading to renku seeing it as a changed input, which would run update again, and so on, without ever stopping.

Due to this, the renku dependency graph has to be *acyclic*. So instead of appending to an input file or writing an output file to the same directory that was used as an input directory, create new files or write to other directories, respectively.

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*
- *dot-full*
- *dot-landscape*
- *dot-full-landscape*
- *dot-debug*
- *json-ld*
- *json-ld-graph*
- *Makefile*
- *nt*
- *rdf*

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
```

Output validation

The `--strict` option forces the output to be validated against the Renku SHACL schema, causing the command to fail if the generated output is not valid, as well as printing detailed information on all the issues found. The `--strict` option is only supported for the `jsonld`, `rdf` and `nt` output formats.

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 save

Convenience method to save local changes and push them to a remote server.

If you have local modification to files, you can save them using

```
$ renku save
Username for 'https://renkulab.io': my.user
Password for 'https://my.user@renkulab.io':
Successfully saved:
    file1
    file2
OK
```

Warning: The username and password for renku save are your gitlab user/password, not your renkulab login!

You can additionally supply a message that describes the changes that you made by using the `-m` or `--message` parameter followed by your message.

```
$ renku save -m "Updated file1 and 2."
Successfully saved:
    file1
    file2
OK
```

If no remote server has been configured, you can specify one by using the `-d` or `--destination` parameter. Otherwise you will get an error.

```
$ renku save
Error: No remote has been set up for the current branch

$ renku save -d https://renkulab.io/gitlab/my.user/my-project.git
Successfully saved:
    file1
    file2
OK
```

You can also specify which paths to save:

```
$ renku save file1
Successfully saved:
    file1
OK
```

4.2.14 `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
$ renku show siblings C G
C
D
---
F
G
$ renku show siblings
A
---
B
---
```

(continues on next page)

(continued from previous page)

```
C
D
---
E
---
F
G
```

You can use the `-f` or `--flat` flag to output a flat list, as well as the `-v` or `--verbose` flag to also output commit information.

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
```

4.2.15 `renku storage`

Manage an external storage.

Pulling files from git LFS

LFS works by checking small pointer files into git and saving the actual contents of a file in LFS. If instead of your file content, you see something like this, it means the file is stored in git LFS and its contents are not currently available locally (they are not pulled):

```
version https://git-lfs.github.com/spec/v1
oid sha256:42b5c7fb2acd54f6d3cd930f18fee3bdcb20598764ca93bdfb38d7989c054bcf
size 12
```

You can manually pull contents of file(s) you want with:

```
$ renku storage pull file1 file2
```

Removing local content of files stored in git LFS

If you want to restore a file back to its pointer file state, for instance to free up space locally, you can run:

```
$ renku storage clean file1 file2
```

This removes any data cached locally for files tracked in in git LFS.

4.2.16 `renku doctor`

Check your system and repository for potential problems.

4.2.17 `renku migrate`

Migrate project to the latest Renku version.

4.2.18 `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".
```

4.2.19 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*, *version='7'*, *agent_version=None*, ***, *client=None*, *creator=None*, *id=None*)

Represent a project.

as_jsonld()
Create JSON-LD.

classmethod from_jsonld(*data*, *client=None*)
Create an instance from JSON-LD data.

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

project_id
Return the id for the project.

to_yaml()
Write an instance to the referenced YAML file.

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*

class renku.core.models.projects.**ProjectSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

Project Schema.

Create an instance.

class Meta

Meta class.

model

alias of *Project*

fix_datetimes(*obj*, *many=False*, ***kwargs*)

Pre dump hook.

renku.core.models.projects.**generate_project_id**(*client*, *name*, *creator*)

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

4.3.2 Datasets

Model objects representing datasets.

Dataset object

class renku.core.models.datasets.**Dataset**(***, *commit=None*, *client=None*, *path=None*, *project: renku.core.models.projects.Project = None*, *parent=None*, *creators*, *id=None*, *label=None*, *date_published=None*, *description=None*, *identifier=NOTHING*, *in_language=None*, *keywords=None*, *license=None*, *title: str = None*, *url=None*, *version=None*, *date_created=NOTHING*, *files=NOTHING*, *tags=NOTHING*, *same_as=None*, *name=None*)

Represent a dataset.

as_jsonld()

Create JSON-LD.

contains_any(*files*)

Check if files are already within a dataset.

creators_csv

Comma-separated list of creators associated with dataset.

creators_full_csv

Comma-separated list of creators with full identity.

data_dir

Directory where dataset files are stored.

default_id()

Configure calculated ID.

default_label()

Generate a default label.

default_reference()

Create a default reference path.

editable
Subset of attributes which user can edit.

entities
Yield itself.

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

find_files(*paths*)
Return all paths that are in files container.

classmethod from_jsonld(*data*, *client=None*, *commit=None*, *schema_class=None*)
Create an instance from JSON-LD data.

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

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

keywords_csv
Comma-separated list of keywords associated with dataset.

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

parent
Return the parent object.

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

set_client(*client*)
Sets the clients on this entity.

short_id
Shorter version of identifier.

submodules
Proxy to client submodules.

tags_csv
Comma-separated list of tags associated with dataset.

to_yaml()
Write 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=None*, *label=NOTHING*, *project: renku.core.models.projects.Project = None*, *parent=None*, *added=NOTHING*, *checksum=None*, *filename=NOTHING*, *name=None*, *filesize=None*, *filetype=None*, *url=None*, *based_on=None*, *external=False*, *source=None*)

Represent a file in a dataset.

as_jsonld()
Create JSON-LD.

default_filename()
Generate default filename based on path.

default_id()
Configure calculated ID.

default_label()
Generate a default label.

default_url()
Generate default url based on project's ID.

entities
Yield itself.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

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

full_path
Return full path in the current reference frame.

parent
Return the parent object.

set_client(*client*)
Sets the clients on this entity.

size_in_mb
Return file size in megabytes.

submodules
Proxy to client submodules.

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: renku.core.models.projects.Project = None*, *id=None*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *generated=None*, *invalidated=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agents=NOTHING*)

Represent an activity in the repository.

as_jsonld()
Create JSON-LD.

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

default_ended_at_time()
Configure calculated properties.

default_generated()
Create default `generated`.

default_id()
Configure calculated ID.

default_influenced()
Calculate default values.

default_invalidated()
Entities invalidated by this Action.

default_label()
Generate a default label.

default_message()
Generate a default message.

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*)
Create an instance from JSON-LD data.

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

classmethod generate_id(*commitsha*)
Calculate action ID.

get_output_paths()
Gets all output paths generated by this run.

nodes
Return topologically sorted nodes.

parents
Return parent commits.

paths
Return all paths in the commit.

removed_paths
Return all paths removed in the commit.

submodules
Proxy to client submodules.

to_yaml()
Write an instance to the referenced YAML file.

class renku.core.models.provenance.activities.**ProcessRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project: renku.core.models.projects.Project = None*, *id=None*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *invalidated=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agents=NOTHING*, *generated=None*, *association=None*, *annotations=None*, *qualified_usage=None*)

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

add_annotations(*annotations*)
Adds annotations from an external tool.

as_jsonld()
Create JSON-LD.

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

default_ended_at_time()
Configure calculated properties.

default_generated()
Create default `generated`.

default_id()
Configure calculated ID.

default_influenced()
Calculate default values.

default_invalidated()
Entities invalidated by this Action.

default_label()
Generate a default label.

default_message()
Generate a default message.

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*)
Create an instance from JSON-LD data.

classmethod from_run(*run*, *client*, *path*, *commit=None*, *subprocess_index=None*, *update_commits=False*)
Convert a `Run` to a `ProcessRun`.

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

classmethod generate_id(*commitsha*)
Calculate action ID.

get_output_paths()
Gets all output paths generated by this run.

nodes
Return topologically sorted nodes.

parents
Return parent commits.

paths
Return all paths in the commit.

plugin_annotations()
Adds `Annotation``s from plugins to a ``ProcessRun`.

removed_paths
Return all paths removed in the commit.

submodules
Proxy to client submodules.

to_yaml()
Write an instance to the referenced YAML file.

class renku.core.models.provenance.activities.**WorkflowRun**(*, *commit=None*, *client=None*, *path=None*, *label=NOTHING*, *project: renku.core.models.projects.Project = None*, *id=None*, *message=NOTHING*, *was_informed_by=NOTHING*, *part_of=None*, *invalidated=None*, *influenced=NOTHING*, *started_at_time=NOTHING*, *ended_at_time=NOTHING*, *agents=NOTHING*, *generated=None*, *association=None*, *annotations=None*, *qualified_usage=None*, *processes=NOTHING*)

A workflow run typically contains several subprocesses.

add_annotations(*annotations*)
Adds annotations from an external tool.

as_jsonld()
Create JSON-LD.

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

default_ended_at_time()
Configure calculated properties.

default_generated()
Create default `generated`.

default_id()
Configure calculated ID.

default_influenced()
Calculate default values.

default_invalidated()
Entities invalidated by this Action.

default_label()
Generate a default label.

default_message()
Generate a default message.

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*)
Create an instance from JSON-LD data.

classmethod from_run(*run*, *client*, *path*, *commit=None*, *update_commits=False*)
Convert a `Run` to a `WorkflowRun`.

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

classmethod generate_id(*commitsha*)
Calculate action ID.

get_output_paths()
Gets all output paths generated by this run.

nodes
Yield all graph nodes.

parents
Return parent commits.

paths
Return all paths in the commit.

plugin_annotations()
Adds `Annotation``s from plugins to a ``ProcessRun`.

removed_paths
Return all paths removed in the commit.

submodules
Proxy to client submodules.

subprocesses
Subprocesses of this `WorkflowRun`.

to_yaml()
Write an instance to the referenced YAML file.

Entities

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

default_id()
Configure calculated ID.

default_label()
Generate a default label.

entities
Yield itself.

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

parent
Return the parent object.

set_client(*client*)
Sets the clients on this entity.

submodules
Proxy to client submodules.

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

default_id()
Configure calculated ID.

default_label()
Generate a default label.

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

entities
Recursively return all files.

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

parent
Return the parent object.

set_client(*client*)
Sets the clients on this entity.

submodules
Proxy to client submodules.

Plans

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

activity
Return the activity object.

default_id()
Configure calculated ID.

default_label()
Generate a default label.

submodules
Proxy to client submodules.

class renku.core.models.provenance.processes.**Workflow**(*, *commit=None*, *client=None*, *path=None*, *id=None*, *label=NOTHING*, *project: renku.core.models.projects.Project = None*, *activity=None*, *subprocesses=NOTHING*)

Represent workflow with subprocesses.

activity
Return the activity object.

default_id()
Configure calculated ID.

default_label()
Generate a default label.

default_subprocesses()
Load subprocesses.

submodules
Proxy to client submodules.

Agents

class renku.core.models.provenance.agents.**Person**(*, *client=None*, *name*, *email=None*, *label=NOTHING*, *affiliation=None*, *alternate_name=None*, *id=None*)

Represent a person.

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

default_id()
Set the default id.

default_label()
Set the default label.

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

classmethod from_dict(*obj*)
Create and instance from a dictionary.

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

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

classmethod from_string(*string*)
Create an instance from a 'Name <email>' string.

full_identity
Return name, email, and affiliation.

short_name
Gives full name in short form.

class renku.core.models.provenance.agents.**SoftwareAgent**(*, *label*, *id*)
Represent executed software.

as_jsonld()
Create JSON-LD.

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

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

Relations

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

as_jsonld()
Create JSON-LD.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

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

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

activity
Return the activity object.

as_jsonld()
Create JSON-LD.

default_id()
Configure calculated ID.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

4.3.4 Renku Workflow

Renku uses PROV-O and its own Renku ontology to represent workflows.

Run

Represents a workflow template.

class renku.core.models.workflow.run.**OrderedSubprocess**(*, *id*, *index: int*, *process*)
A subprocess with ordering.

static generate_id(*parent_id*, *index*)
Generate an id for an `OrderedSubprocess`.

class renku.core.models.workflow.run.**OrderedSubprocessSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

OrderedSubprocess schema.

Create an instance.

class Meta

Meta class.

model

alias of *OrderedSubprocess*

class renku.core.models.workflow.run.**Run**(*, *commit=None*, *client=None*, *path=None*, *id=None*, *label=NOTHING*, *project: renku.core.models.projects.Project = None*, *command: str = None*, *successcodes: list = NOTHING*, *subprocesses=NOTHING*, *arguments=NOTHING*, *inputs=NOTHING*, *outputs=NOTHING*)

Represents a *renku run* execution template.

activity

Return the activity object.

add_subprocess(*subprocess*)

Adds a subprocess to this run.

as_jsonld()

Create JSON-LD.

classmethod from_factory(*factory*, *client*, *commit*, *path*)

Creates a `Run` from a `CommandLineToolFactory`.

classmethod from_jsonld(*data*)

Create an instance from JSON-LD data.

static generate_id(*client*)

Generate an id for an argument.

to_argv()

Convert run into argv list.

to_stream_repr()

Input/output stream representation.

update_id_and_label_from_commit_path(*client*, *commit*, *path*, *is_subprocess=False*)

Updates the _id and _label using supplied commit and path.

class renku.core.models.workflow.run.**RunSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

Run schema.

Create an instance.

class Meta

Meta class.

model

alias of *Run*

Parameters

Represents a workflow template.

class renku.core.models.workflow.parameters.**CommandArgument**(*, *id=None*, *label=None*, *position: int = None*, *prefix: str = None*, *value: str = None*)

An argument to a command that is neither input nor output.

as_jsonld()
Create JSON-LD.

default_label()
Set default label.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

static generate_id(*run_id*, *position=None*)
Generate an id for an argument.

to_argv()
String representation (sames as cmd argument).

class renku.core.models.workflow.parameters.**CommandArgumentSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

CommandArgument schema.

Create an instance.

class Meta
Meta class.

model
alias of *CommandArgument*

class renku.core.models.workflow.parameters.**CommandInput**(*, *id=None*, *label=None*, *position: int = None*, *prefix: str = None*, *consumes*, *mapped_to=None*)

An input to a command.

as_jsonld()
Create JSON-LD.

default_label()
Set default label.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

static generate_id(*run_id*, *position=None*)
Generate an id for an argument.

to_argv()
String representation (sames as cmd argument).

to_stream_repr()
Input stream representation.

class renku.core.models.workflow.parameters.**CommandInputSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)
CommandArgument schema.

Create an instance.

class Meta
Meta class.

model
alias of *CommandInput*

class renku.core.models.workflow.parameters.**CommandOutput**(*, *id=None*, *label=None*, *position: int = None*, *prefix: str = None*, *create_folder: bool = False*, *produces*, *mapped_to=None*)
An output of a command.

as_jsonld()
Create JSON-LD.

default_label()
Set default label.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

static generate_id(*run_id*, *position=None*)
Generate an id for an argument.

to_argv()
String representation (sames as cmd argument).

to_stream_repr()
Input stream representation.

class renku.core.models.workflow.parameters.**CommandOutputSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)
CommandArgument schema.

Create an instance.

class Meta
Meta class.

model
alias of *CommandOutput*

class renku.core.models.workflow.parameters.**CommandParameter**(*, *id=None*, *label=None*, *position: int = None*, *prefix: str = None*)
Represents a parameter for an execution template.

sanitized_id
Return _id sanitized for use in non-jsonld contexts.

class renku.core.models.workflow.parameters.**CommandParameterSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

CommandParameter schema.

Create an instance.

class Meta
Meta class.

model
alias of *CommandParameter*

class renku.core.models.workflow.parameters.**MappedIOStream**(*, *client=None*, *id=None*, *label=None*, *stream_type: str*)

Represents an IO stream (stdin, stdout, stderr).

as_jsonld()
Create JSON-LD.

default_id()
Generate an id for a mapped stream.

default_label()
Set default label.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

class renku.core.models.workflow.parameters.**MappedIOStreamSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)

MappedIOStream schema.

Create an instance.

class Meta
Meta class.

model
alias of *MappedIOStream*

4.3.5 Renku Workflow Conversion

Renku allows conversion of tracked workflows to runnable workflows in supported tools (Currently CWL)

CWL

Converter for workflows to cwl.

class renku.core.models.workflow.converters.cwl.**CWLConverter**
Converts a Run to cwl file(s).

static convert(*run*, *client*, *path=None*)
Convert the workflow to one ore more .cwl files.

4.3.6 Tools and Workflows

Manage creation of tools and workflows for workflow tracking.

Command-line tool

Represent a `CommandLineToolFactory` for tracking workflows.

class renku.core.models.cwl.command_line_tool.**CommandLineToolFactory**(*command_line*, *explicit_inputs=NOTHING*, *explicit_outputs=NOTHING*, *no_input_detection=False*, *no_output_detection=False*, *directory='.'*, *working_dir='.'*, *stdin=None*, *stderr=None*, *stdout=None*, *successCodes=NOTHING*, *annotations=None*, *messages=None*, *warnings=None*)

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.

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

generate_process_run(*client*, *commit*, *path*)
Return an instance of `ProcessRun`.

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

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

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

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

iter_input_files(*basedir*)
Yield tuples with input id and path.

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.

Annotation

Represent an annotation for a workflow.

class renku.core.models.cwl.annotation.**Annotation**(*, *id*, *body=None*, *source=None*)
Represents a custom annotation for a research object.

as_jsonld()
Create JSON-LD.

classmethod from_jsonld(*data*)
Create an instance from JSON-LD data.

class renku.core.models.cwl.annotation.**AnnotationSchema**(**args*, *commit=None*, *client=None*, ***kwargs*)
Annotation schema.

Create an instance.

class Meta
Meta class.

model
alias of *Annotation*

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.

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**(*steps=NOTHING*)
Define a workflow representation.

add_step(***kwargs*)
Add a workflow step.

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.

4.3.7 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*, *no_slashes=False*)
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*, *external_storage_requested=True*, ***, *data_dir='data'*)
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**
Client for handling datasets.

CACHE = 'cache'
Directory to cache transient data.

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

POINTERS = 'pointers'
Directory for storing external pointer files.

add_data_to_dataset(*dataset*, *urls*, *force=False*, *overwrite=False*, *sources=()*, *destination=''*, *ref=None*, *external=False*, *extract=False*, *all_at_once=False*, *destination_names=None*, *progress=None*)
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 errors.ParameterError

create_dataset(*name=None*, *title=None*, *description=None*, *creators=None*, *keywords=None*)
Create a dataset.

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

Commits are returned sorted from newest to oldest.

datasets
Return mapping from path to dataset.

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

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

has_external_files()
Return True if project has external files.

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

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

prepare_git_repo(*url*, *ref=None*)
Clone and cache a Git repo.

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

remove_file(*filepath*)
Remove a file/symlink and its pointer file (for external files).

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

renku_pointers_path
Return a `Path` instance of Renku pointer files folder.

update_dataset_files(*files*, *ref*, *delete=False*)
Update files and dataset metadata according to their remotes.

Parameters

- **files** – List of files to be updated
- **delete** – Indicates whether to delete files or not

Returns List of files that should be deleted

update_external_files(*records*)
Update files linked to external storage.

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

class renku.core.management.datasets.**DownloadProgressCallback**(*description*, *total_size*)
Interface to report various stages of a download.

Default initializer.

finalize()
Called once when the download is finished.

update(*size*)
Update the status.

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*, ***, *data_dir='data'*)
Client for handling a local repository.

ACTIVITY_INDEX = 'activity_index.yaml'
Caches activities that generated a path.

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.

activities_for_paths(*paths*, *file_commit=None*, *revision='HEAD'*)
Get all activities involving a path.

activity_index_path
Path to the activity filepath cache.

add_to_activity_index(*activity*)
Add an activity and it's generations to the cache.

cwl_prefix
Return a CWL prefix.

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

find_previous_commit(*paths*, *revision='HEAD'*, *return_first=False*, *full=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
- **full** – return full history

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

import_from_template(*template_path*, *metadata*, *force=False*)
Render template files from a template directory.

init_repository(*force=False*, *user=None*)
Initialize an empty Renku repository.

is_project_set()
Return if project is set for the client.

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

latest_agent
Returns latest agent version used in the repository.

lock
Create a Renku config lock.

parent = None
Store a pointer to the parent repository.

path_activity_cache
Cache of all activities and their generated paths.

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 Project instance.

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*, *name=None*)
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.

renku.core.management.repository.**path_converter**(*path*)
Converter for path in PathMixin.

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(*commit_only=None*, *commit_empty=True*, *raise_if_empty=False*, *commit_message=None*)
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.

setup_credential_helper()
Setup git credential helper to `cache` if not set already.

transaction(*clean=True*, *commit=True*, *commit_empty=True*, *commit_message=None*, *commit_only=None*, *ignore_std_streams=False*, *raise_if_empty=False*)
Perform Git checks and operations.

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

renku.core.management.git.**get_mapped_std_streams**(*lookup_paths*, *streams=('stdin', 'stdout', 'stderr')*)
Get a mapping of standard streams to given paths.

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.

renku.core.models.git.**get_user_info**(*git*)
Get Git repository's owner name and email.

4.5 Plugin Support

Plugins are supported using the pluggy library.

Plugins can be created as python packages that contain the respective entrypoint definition in their *setup.py* file, like so:

```
from setuptools import setup

setup(
    ...
    entry_points={"renku": ["name_of_plugin = myproject.pluginmodule"]},
    ...
)
```

where *myproject.pluginmodule* points to a Renku *hookimpl* e.g.:

```
from renku.core.plugins import hookimpl

@hookimpl
def plugin_hook_implementation(param1, param2):
    ...
```

4.5.1 `renku run` hooks

Plugin hooks for renku run customization.

`renku.core.plugins.run.`**`cmdline_tool_annotations`**(*tool*)
Plugin Hook to add `Annotation` entry list to a `WorkflowTool`.

Parameters **`run`** – A `WorkflowTool` object to get annotations for.

Returns A list of `renku.core.models.cwl.annotation.Annotation` objects.

`renku.core.plugins.run.`**`pre_run`**(*tool*)
Plugin Hook that gets called at the start of a `renku run` call.

Can be used to setup plugins that get executed during the run.

Parameters **`run`** – A `WorkflowTool` object that will get executed by `renku run`.

`renku.core.plugins.run.`**`process_run_annotations`**(*run*)
Plugin Hook to add `Annotation` entry list to a `ProcessRun`.

Parameters **`run`** – A `ProcessRun` object to get annotations for.

Returns A list of `renku.core.models.cwl.annotation.Annotation` objects.

Python Module Index

r

Index

A

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W