cenv

Release 2019

Simon Kallfass

Nov 24, 2019

CONTENTS

Tool to create / update conda environments from `meta.yaml`.

Due to the redundant dependency information inside the `meta.yaml` (required to create the conda-package) and the `environment.yml` (as definition file for the conda-environment during development and for production), `cenv` (short form for `conda-env-manager`) was created to make the `meta.yaml` the only relevant file and to create and update conda-environment from the definition inside this `meta.yaml`. The name of the conda-environment to create / update is defined in the section `extra:cenv` and the variable `env_name` inside the `meta.yaml` (at `conda-build/meta.yaml`). The python version must be defined in `extra:cenv` inside the key `python`.

The steps run by cenv:

- creation of a backup if the environment already exists followed by the removal of the previous environment.
- creation of the environment as defined in the `meta.yaml`. If any failures occurred during creation and the backup was created, the command to reset the backup-version can be used.
- if enabled in the config file the environment.yml is exported after creation / update of the environment.

The usage of cenv reduces the conda commands to use to the following:

- `conda activate ...` to activate the environment
- `conda deactivate` to deactivate an environment
- `conda info` to show information about the currently activated environment
- `conda search ...` to search for availability of a package in the conda channels.
- `conda remove -n ... --all` to remove an environment
- `cenv` to create / update an environment

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

To install cenv simply run:

```
pip install cenv_tool
```

Now run *init_cenv* to create the relevant config-files and add the autoactivate- and autoupdate-shell-function to your *.bashrc* / *.zshrc*.

1.1.1 autoactivate and autoupdate

Per default these features are deactivated, even if added to your shell by running *init_cenv*.

autoactivate-feature

The autoactivate-feature activates the conda-environment as named *extra*-section in the meta.yaml located at *conda-build/meta.yaml*, if the environment exists. To activate the autoactivate-features run:

```
autoactivate_toggle
```

autoupdate-feature

The autoupdate checks if the content of the meta.yaml changed. The current state is stored as a md5sum in *conda-build/meta.md5*. If it changed the cenv-process is called.

For the autoupdate-feature run:

```
autoupdate_toggle
```

1.2 Configuration

cenv uses the path */opt/conda* as default conda-installation-folder and */shared/conda/envs* as default conda-environments-folder.

You can overwrite these settings with a *cenv.yml* at *~/.config/cenv/cenv.yml* with the following content:

```
conda_folder: /opt/conda
env_folder: /shared/conda/envs
export_environment_yml: false
```

There you can define your own conda-installation-path and the conda-environments-folder. The functionality to export the created / updated environment into a *environment.yml* can be activated / deactivated here, too. Per default it is deactivated. If this is activated, the environment.yml will be placed at *conda-build/environment.yml*.

1.3 Usage

All steps required to create or update the projects conda environment are run automatically running `cenv` inside the project folder:

Attention: If you use cenv, each environment should only be created, updated and modified using *cenv*! This means the commands *conda install*, *conda remove* are not used anymore. Changes of the dependencies of the environment are defined inside the *meta.yaml* and are applied by using *cenv*.

This means:

- new dependency required => add it in *meta.yaml* and run *cenv*.
- dependency not needed anymore => remove it from *meta.yaml* and run *cenv*.
- need of another version of dependency => change the version of dependency in *meta.yaml* and run *cenv*.

1.3.1 Project structure

A project using cenv needs at minimum the following folder structure:

```
<PROJECT>
  ├── conda-build
  │   └── meta.yaml
  ├── <SOURCE_CODE>
  ├── README.md
  └── setup.py
```

1.3.2 meta.yaml

The required information about the projects conda environment are extracted from the meta.yaml. This meta.yaml should be located inside the project folder at `./conda-build/meta.yaml`.

The project-configuration is defined in the `extra` section of the `meta.yaml`. There you can define the name of the projects conda-environment at `extra:cenv:env_name`. The python version has to be defined here at `extra:cenv:python`, too. Also you can define requirements only needed during development but not to be included into the resulting conda package. These requirements have to be defined in the `extra:cenv:dev_requirements`-section.

All other parts of the `meta.yaml` have to be defined as default.

A meta.yaml valid for cenv should look like the following:

```
{% set data = load_setup_py_data() %}

package:
    name: "example_package"
    version: {{ data.get("version") }}

source:
    path: ..

build:
    build: {{ environ.get('GIT_DESCRIBE_NUMBER', 0) }}
    preserve_egg_dir: True
    script: python -m pip install --no-deps --ignore-installed .

requirements:
    build:
        - python
        - pip
        - setuptools
    run:
        - python
        - attrs >=18.2,<19
        - jinja2 >=2.10
        - six >=1.12.0
        - yaml >=0.1.7
    run_constrained:
        - pandas >=0.23

test:
    imports:
        - example_package

extra:
    cenv:
        env_name: example
        python: 3.6.8
        dev_requirements:
            - ipython >=7
```

Attention: In the `requirements:run`-section the minimal version of each package has to be defined like the following:

```
- package >=0.1
```

The same is required for the `extra:cenv:dev_requirements`-section. If the section `requirements:run_constrained` is defined, too, these dependency information is extracted for dependency collection, too. Not defining a version will not create or update a conda-environment, because this is not the purpose of the conda-usage. The validity of the `meta.yaml` is checked in `cenv` using the *marshmallow* package. You can additionally add upper limits for the version like the following:

```
- package >=0.1,<0.3
```

If cenv is run the environment is created / updated from the definition inside this `meta.yaml`. The creation of the backup of the previous environment ensures to undo changes if any error occurs during recreation of the environment.

Attention: `cenv` can only update the environment if it is not activated. So ensure the environment to be deactivated before running `cenv`.

Per default exporting the conda environment definition into an environment.yml is turned off. If you want to turn this functionality on you need to modify your `~/.config/cenv.yml` as described in configuration.

1.3.3 Running cenv

Example for the output of the `cenv` command:

On create:

```
Creating cenv_dev
    ├── Create environment
    │   └── Created
    ├── write md5sum of meta.yaml
    │   └── updated
    └── Done
```

On update:

```
Updating cenv_dev
    ├── Create backup
    │   └── Created
    ├── Remove existing env
    │   └── Removed
    ├── Create environment
    │   ├── Clear backup
    │   │   └── Cleared
    │   └── Created
    ├── write md5sum of meta.yaml
    │   └── updated
    └── Done
```

1.4 Development of cenv

1.4.1 Develop cenv

To create / update the dev environment to develop cenv run the pre-commit hooks manually:

```
pyenv local 3.7.3
dephell venv shell --env=dev
dephell deps install
pre-commit run --all-files
```

1.4.2 Running tests

To create / update the test environment run:

```
dephell venv shell --env=pytest
dephell deps install
```

To run all tests run the following command:

```
dephell project test --env=pytest
```

1.4.3 Updating the docs

To create / update the docs environment run:

```
dephell venv shell --env=docs
dephell deps install --env=docs
```

To create / update the docs first run the tests as described above. Then run:

```
dephell venv shell --env=docs
sphinx-apidoc -f -o docs cenv_tool && sphinx-build -W docs docs/build
```

1.5 Code documentation

1.5.1 cenv_tool

cenv_tool package

Submodules

cenv_tool.init_cenv module

Install config and cenv.sh.

cenv_tool.init_cenv.**initialize_cenv**(*config_path*, *autoenv_script_path*, *autoenv_script_source_path*, *config_file*, *config_file_source*, *zshrc*, *bashrc*)

Install user-config and cenv.sh for autoactivate and autoupdate.

Parameters

- **config_path** (Path) – the path for cenv config-stuff.
- **autoenv_script_path** (Path) – the path to install the cenv.sh script to.
- **autoenv_script_source_path** (Path) – the path where to get the cenv.sh script from
- **config_file** (Path) – the path to install the user-config into.
- **config_file_source** (Path) – the path where to get the config file from.
- **zshrc** (Path) – the path to the users .zshrc
- **bashrc** (Path) – the path to the users .bashrc

Return type NoReturn

cenv_tool.init_cenv.**main**()

Call the initialization function to install config and cenv.sh.

cenv_tool.project module

Contain the logic for conda environment creation from `meta.yaml`.

cenv is a tool to handle conda environment creation and update from the dependency-definition inside the `meta.yaml` file.

As default conda has two files for dependency management: * the `environment.yml` * and the `meta.yaml`

In the `environment.yml` the environment-definition is stored. In the `meta.yaml` the required information to build a conda-package are stored. This means redundant information.

cenv collects the dependency-information and all project-specific settings from the `meta.yaml`.

The collected information is used to create / update the projects conda environment.

class cenv_tool.project.**Project**(*rules*, *conda_folder=None*, *env_folder=None*, *env_name=None*, *dependencies=None*, *is_env=None*, *export_environment_yml=None*, *cmds=None*, *cmd_kwargs=None*, *is_git=None*)

Bases: `object`

Contain a python-project using conda environments.

Containing methods to display information to current project and methods to update the projects conda-environment from the settings defined in the projects `meta.yaml`.

__attrs_post_init__()
Set the more complex attributes of the project class.

_handle_existing_environment()
Check if environment already exists and create a backup of it.

Return type `bool`

_remove_backup_environment()
Remove backup environment cloned from original environment.

Return type `NoReturn`

_remove_previous_environment()
Remove old version of project environment.

If the old environment can't be removed, the backup made is removed.

Return type `NoReturn`

_restore_environment_from_backup(*cloned*)
Restore the environment from the cloned backup environment.

After restore the backup environment is removed.

Parameters **cloned** (`bool`) – indicates if the environment already existed and a backup was created.

Return type `NoReturn`

clone_environment_as_backup()
Clone the existing environment as a backup.

If the backup already exists, the previous backup is removed, then the new one is created by cloning the current project environment.

Return type `NoReturn`

cmd_kwargs

cmds

collect_available_envs()
Collect the names of the conda environments currently installed.

Parameters **conda_folder** – the path where conda is installed.

Return type List[str]

Returns list of currently installed conda-environments

conda_folder

create_environment(*cloned*)
Create the environment for the project.

Try to create the environment for the project. If the environment already existed and a backup was made and any error occure, restore the backup environment. If everything worked correctly finally remove the backup (if one was made).

Parameters **cloned** (bool) – indicates if the environment already existed and a backup was created.

Return type NoReturn

dependencies

env_folder

env_name

export_environment_definition()
Export projects environment definition to an environment.yml.

Return type NoReturn

export_environment_yml

is_env

is_git

rules

update()
Create / recreate the conda environment of the current project.

If the conda environment already exists, clone the environment as a backup and then remove original environment. Then create the new conda environment. If a backup was created it is removed afterwards. If any errors occurs during creation of the new environment, recreate the old environment from backup and remove the backup afterwards. If activated in the config-file, export the environment-definition of the created environment to an environment.yml file. Finally store the md5sum of the meta.yaml for the autoupdate feature.

Return type NoReturn

write_new_md5sum()
Write new md5sum of meta.yaml to conda-build/meta.md5.

cenv_tool.project.**_build_arguments**()
Create arguments for the cenv-tool.

Return type ArgumentParser

Returns the parsed arguments.

cenv_tool.project.**main**()
Collect the required args, initialize and run the Project.

Return type NoReturn

cenv_tool.rules module

Rules-definitions required by cenv.

class cenv_tool.rules.**CondaCmdFormats**(*remove='{conda} remove -n {name} –all -y'*, *export='{conda} env export -n {name} > conda-build/environment.yml'*, *create='{conda} create -n {name} {pkgs} -y'*, *clone='{conda} create -n {name}_backup –clone {name} -y'*, *restore='{conda} create -n {name} –clone {name}_backup -y'*, *clean='{conda} remove -n {name}_backup –all -y'*)
Bases: object

Contain the formats for the conda commands to use inside cenv.

Variables

- ***remove*** – command to remove a conda environment.
- ***export*** – command to use to export a conda environment to an environment definition file (environment.yml).
- ***create*** – command to use for conda environment creation.
- ***clone*** – command to use to clone a conda environment.
- ***restore*** – command to use to recreate a conda environment from backup conda environment (clone).
- ***clean*** – command to use to remove the backup conda environment.

clean

clone

conda_bin(*conda_folder*)
Combine the path of conda-folder with subpath of conda-bin.

Returns the path to the conda-executable

create

export

remove

restore

class cenv_tool.rules.**Rules**
Bases: object

Contain the rules required by cenv-tool.

conda_cmds = CondaCmdFormats(remove='{conda} remove -n {name} --all -y', export='{conda

git_folder = '.git'

cenv_tool.schemata module

Contain schemata required by cenv-tool.

class cenv_tool.schemata.**SMetaYaml**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain the representable of a complete meta.yaml file.

Schema for a meta.yaml file to be used for cenv. Ensure the meta.yaml to load contains the relevant information about the package, source, build, requirements and extra. The test-section is optional.

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNBuild**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain the build-section inside a meta.yaml.

The build-section requires to define the build-number, if the egg-dir should be preserved, the script to run on installation and if any entrypoints are defined for the package.

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNCenv**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNExtra**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain the extra-section inside a meta.yaml.

The extra-section has to contains the information where to find the conda-folder, the name of the conda environment to use for the current project and the cenv-version used when the meta.yaml file was created.

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNPackage**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain the package-section inside a meta.yaml.

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNRequirements**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain requirements-section inside a meta.yaml.

The underlying build- and run-sections have to be valid!

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNSource**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain the source-section inside a meta.yaml.

opts = <marshmallow.schema.SchemaOpts object>

class cenv_tool.schemata.**SNTest**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: marshmallow.schema.Schema

Contain tests-section inside a meta.yaml.

opts = <marshmallow.schema.SchemaOpts object>

cenv_tool.utils module

Contain utils required by cenv-tool.

exception cenv_tool.utils.**CenvProcessError**

Bases: Exception

Represent a process error during cenv execution.

class cenv_tool.utils.**_NullUndefined**(*hint=None*, *obj=missing*, *name=None*, *exc=<class 'jinja2.exceptions.UndefinedError'>*)

Bases: jinja2.runtime.Undefined

Handle jinja2-variables with undefined content of meta.yaml.

__getattr__(*attribute_name*)

Replace getattr dunder of this class.

__getitem__(*attribute_name*)

Replace getitem dunder of this class.

__unicode__()

Replace unicode dunder of this class.

class cenv_tool.utils.**_StrDict**

Bases: dict

Handle dictionaries for jinja2-variables of meta.yaml.

__getitem__(*key*, *default=''*)

Replace getitem dunder of this class.

Return type str

cenv_tool.utils.**extract_dependencies_from_meta_yaml**(*meta_yaml_content*)

Extract the dependencies defined in the requirements-run-section.

If additional dev-requirements are defined in the extra-dev_requirements-section, these dependencies are added to the other dependencies.

Parameters **meta_yaml_content** (dict) – the content from a meta.yaml as a dict.

Return type List[str]

Returns the collected dependencies.

cenv_tool.utils.**message**(*, *text*, *color*, *special=None*, *indent=1*)
Print passed `text` in the passed `color` on terminal.

Parameters

- **text** (`str`) – the text to print colored on terminal.
- **color** (`str`) – the color of the text to print.
- **special** (`Optional[str]`) – special kind of message to print. Available are `'row'` and `'end'`.
- **indent** (`int`) – the indent to use for the text.

Return type `NoReturn`

cenv_tool.utils.**read_config**()
Read the config file for cenv from the users-home path if it exists.

If there is no user-config-file the default one is used.

Returns the content of the read config file.

cenv_tool.utils.**read_meta_yaml**(*path*)
Read the meta.yaml file.

The file is read from relative path `conda-build/meta.yaml` inside the current path, validate the `meta.yaml` using the marshmallow-schema, `SMetaYaml`, extract the project-settings.

Parameters **path** (`Path`) – the current working directory.

Return type `dict`

Returns the `meta.yaml` content as a dict.

cenv_tool.utils.**run_in_bash**(*cmd*)
Run passed `cmd` inside bash using `subprocess.check_output()`.

Parameters **cmd** (`str`) – the command to execute.

Return type `str`

Returns the output of the ran command.

Module contents

Conda environment creation and update from meta.yaml.

1.6 About

- **Author**: Simon Kallfass
- **Homepage**: https://www.ouroboros.info
- **Email**: skallfass@ouroboros.info

1.7 License

1.8 Impressum

1.8.1 Legal Disclosure

Information in accordance with section 5 TMG

- Simon Kallfass
- Hofäckerstr. 46
- 76139 Karlsruhe

Contact

- Telephone: +49 177 176 7126
- E-Mail: skallfass@ouroboros.info
- Homepage: https://www.ouroboros.info

Disclaimer

Accountability for content The contents of our pages have been created with the utmost care. However, we cannot guarantee the contents' accuracy, completeness or topicality. According to statutory provisions, we are furthermore responsible for our own content on these web pages. In this context, please note that we are accordingly not obliged to monitor merely the transmitted or saved information of third parties, or investigate circumstances pointing to illegal activity. Our obligations to remove or block the use of information under generally applicable laws remain unaffected by this as per §§ 8 to 10 of the Telemedia Act (TMG).

Accountability for links

Responsibility for the content of external links (to web pages of third parties) lies solely with the operators of the linked pages. No violations were evident to us at the time of linking. Should any legal infringement become known to us, we will remove the respective link immediately.

Copyright

Our web pages and their contents are subject to German copyright law. Unless expressly permitted by law (§ 44a et seq. of the copyright law), every form of utilizing, reproducing or processing works subject to copyright protection on our web pages requires the prior consent of the respective owner of the rights. Individual reproductions of a work are allowed only for private use, so must not serve either directly or indirectly for earnings. Unauthorized utilization of copyrighted works is punishable (§ 106 of the copyright law).

1.8.2 Privacy Statement

General

Your personal data (e.g. title, name, house address, e-mail address, phone number, bank details, credit card number) are processed by us only in accordance with the provisions of German data privacy laws. The following provisions describe the type, scope and purpose of collecting, processing and utilizing personal data. This data privacy policy applies only to our web pages. If links on our pages route you to other pages, please inquire there about how your data are handled in such cases.

Inventory data

1. Your personal data, insofar as these are necessary for this contractual relationship (inventory data) in terms of its establishment, organization of content and modifications, are used exclusively for fulfilling the contract. For goods to be delivered, for instance, your name and address must be relayed to the supplier of the goods.
2. Without your explicit consent or a legal basis, your personal data are not passed on to third parties outside the scope of fulfilling this contract. After completion of the contract, your data are blocked against further use. After expiry of deadlines as per tax-related and commercial regulations, these data are deleted unless you have expressly consented to their further use.

Disclosure

According to the Federal Data Protection Act, you have a right to free-of-charge information about your stored data, and possibly entitlement to correction, blocking or deletion of such data. Inquiries can be directed to the following e-mail addresses: skallfass@ouroboros.info

PYTHON MODULE INDEX

C

INDEX

Symbols

C

D

E

G

I

M

O

P

R

S

U

W