RiotKit Do Documentation

Release 1

Wolnosciowiec Team

Jul 13, 2020

Contents:

Stop writing hacks in Makefile, use Python snippets for advanced usage, for the rest use simple few lines of Bash, share code between your projects using Python Packages.

RKD can be used on PRODUCTION, for development, for testing, to replace some of Bash scripts inside docker containers, and for many more, where Makefile was used.

CHAPTER 1

Example use cases

- Docker based production environment with multiple configuration files, procedures (see: Harbor project)
- Database administrator workspace (importing dumps, creating new user accounts, plugging/unplugging databases)
- Development environment (executing migrations, importing test database, splitting tests and running parallel)
- On CI (prepare project to run on eg. Jenkins or Gitlab CI) - RKD is reproducible on local computer which makes inspection easier
- Kubernetes/OKD deployment workspace (create shared YAML parts with JINJA2 between multiple environments and deploy from RKD)
- Automate things like certificate regeneration on production server, RKD can generate any application configs using JINJA2
- Installers (RKD has built-in commands for replacing lines in files, modifying .env files)

CHAPTER 2

Quick start

```
# 1) via PIP
pip install rkd

# 2) Create project (will create a virtual env and commit files to GIT)
rkd :rkd:create-structure --commit
```

CHAPTER 3

Getting started with RKD

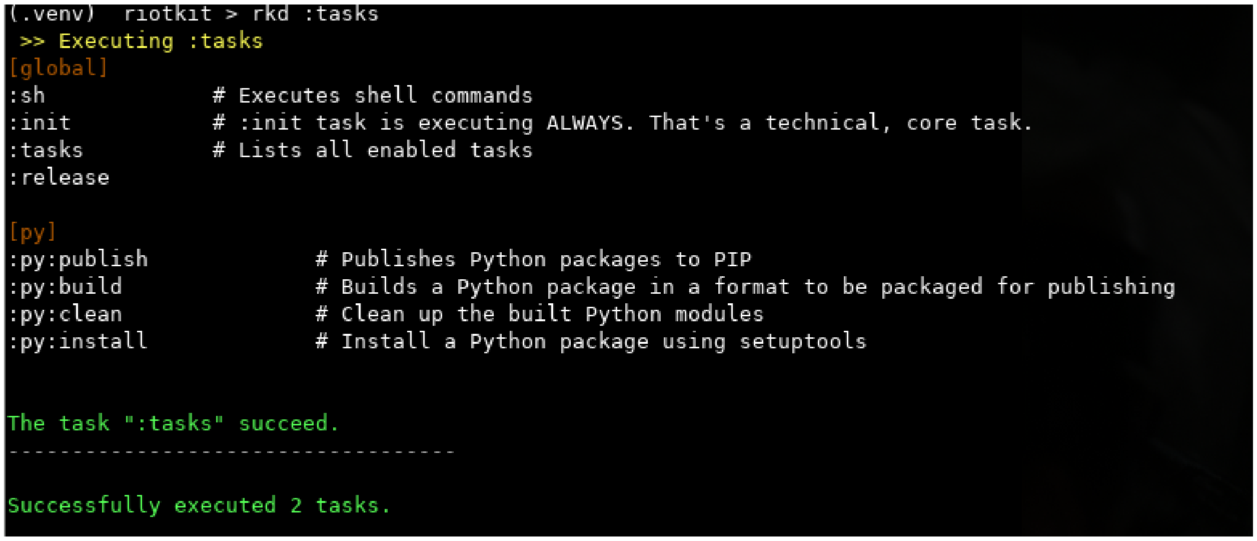
The “Quick start” section will end up with a **.rkd** directory, a requirements.txt and setup-venv.sh

1. Use setup-venv.sh to enter shell of your project, where RKD is installed with all dependencies
2. Each time you install anything from **pip** in your project - add it to requirements.txt, you can install additional RKD tasks from pip
3. In **.rkd/makefile.yaml** you can start adding your first tasks and imports

CHAPTER 4

Read more

- YAML syntax is described in *Tasks development* section
- Writing Python code in makefile.yaml requires to lookup *Tasks API*
- Learn how to import installed tasks via pip - *Importing tasks*

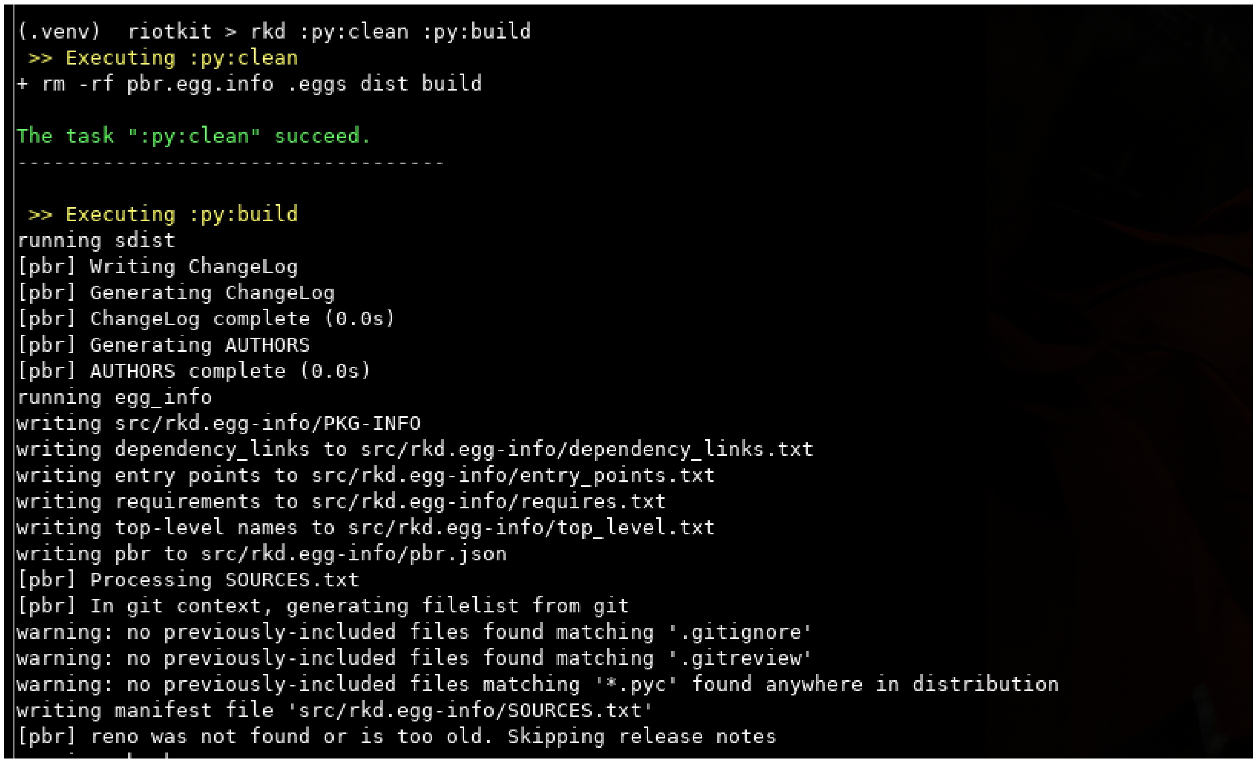


```
(.venv)  riotkit > rkd :tasks
 >> Executing :tasks
[global]
:sh              # Executes shell commands
:init            # :init task is executing ALWAYS. That's a technical, core task.
:tasks           # Lists all enabled tasks
:release

[py]
:py:publish              # Publishes Python packages to PIP
:py:build                # Builds a Python package in a format to be packaged for publishing
:py:clean                # Clean up the built Python modules
:py:install              # Install a Python package using setuptools


The task ":tasks" succeed.
-----------------------------------

Successfully executed 2 tasks.
```



```
(.venv)  riotkit > rkd :py:clean :py:build
 >> Executing :py:clean
+ rm -rf pbr.egg.info .eggs dist build

The task ":py:clean" succeed.
-----------------------------------

 >> Executing :py:build
running sdist
[pbr] Writing ChangeLog
[pbr] Generating ChangeLog
[pbr] ChangeLog complete (0.0s)
[pbr] Generating AUTHORS
[pbr] AUTHORS complete (0.0s)
running egg_info
writing src/rkd.egg-info/PKG-INFO
writing dependency_links to src/rkd.egg-info/dependency_links.txt
writing entry points to src/rkd.egg-info/entry_points.txt
writing requirements to src/rkd.egg-info/requires.txt
writing top-level names to src/rkd.egg-info/top_level.txt
writing pbr to src/rkd.egg-info/pbr.json
[pbr] Processing SOURCES.txt
[pbr] In git context, generating filelist from git
warning: no previously-included files found matching '.gitignore'
warning: no previously-included files found matching '.gitreview'
warning: no previously-included files matching '*.pyc' found anywhere in distribution
writing manifest file 'src/rkd.egg-info/SOURCES.txt'
[pbr] reno was not found or is too old. Skipping release notes
```

4.1 Basics

RKD command-line usage is highly inspired by GNU Make and Gradle, but it has its own extended possibilities to make your scripts smaller and more readable.

- Tasks are prefixed always with ":".
- Each task can handle it's own arguments (unique in RKD)
- "@" allows to propagate arguments to next tasks (unique in RKD)

4.1.1 Tasks arguments usage in shell and in scripts

Executing multiple tasks in one command:

```
rkd :task1 :task2
```

Multiple tasks with different switches:

```
rkd :task1 --hello  :task2 --world
```

Tasks sharing the same switches

Both tasks will receive switch "–hello"

```
# expands to:
#  :task1 --hello
#  :task2 --hello
rkd @ --hello :task1 :task2
```

(continues on next page)

(continued from previous page)

```
# handy, huh?
```

Advanced usage of shared switches

Operator "@" can set switches anytime, it can also clear or replace switches in **NEXT TASKS**.

```
# expands to:
#   :task1 --hello
#   :task2 --hello
#   :task3
#   :task4 --world
#   :task5 --world
rkd @ --hello :task1 :task2 @ :task3 @ --world :task4 :task5
```

Written as a pipeline (regular bash syntax)

It's exactly the same example as above, but written multiline. It's recommended to write multiline commands if they are longer.

```
rkd @ --hello \
    :task1 \
    :task2 \
    @
    :task3 \
    @ --world \
    :task4 \
    :task5
```

4.1.2 YAML syntax - makefile.yaml

YAML syntax has an advantage of simplicity and clean syntax, custom bash tasks can be defined there easier than in Python. To use YAML you need to define **makefile.yaml** file in .rkd directory.

NOTICE: makefile.py and makefile.yaml can exist together. Python version will be loaded first, the YAML version will append changes in priority.

```
version: org.riotkit.rkd/0.3
imports:
  - rkd.standardlib.docker.TagImageTask

tasks:
  # see this task in "rkd :tasks"
  # run with "rkd :examples:bash-test"
  :examples:bash-test:
      description: Execute an example command in bash - show only python related tasks
      steps: |
             echo "RKD_DEPTH: ${RKD_DEPTH} # >= 2 means we are running rkd-in-rkd"
             echo "RKD_PATH: ${RKD_PATH}"
             rkd --silent :tasks | grep ":py"

  # try "rkd :examples:arguments-test --text=Hello --test-boolean"
  :examples:arguments-test:
      description: Show example usage of arguments in Bash
      arguments:
          "--text":
```

(continues on next page)

(continued from previous page)

```
            help: "Adds text message"
            required: True
        "--test-boolean":
            help: "Example of a boolean flag"
            action: store_true # or store_false
    steps:
      - |
        #!bash
        echo " ==> In Bash"
        echo " Text: ${ARG_TEXT}"
        echo " Boolean test: ${ARG_TEST_BOOLEAN}"
      - |
        #!python
        print(' ==> In Python')
        print(' Text: %s ' % ctx.args['text'])
        print(' Text: %s ' % str(ctx.args['test_boolean']))
        return True

# run with "rkd :examples:list-standardlib-modules"
:examples:list-standardlib-modules:
    description: List all modules in the standardlib
    steps:
      - |
        #!python
        ctx: ExecutionContext
        this: TaskInterface

        import os

        print('Hello world')
        print(os)
        print(ctx)
        print(this)

        return True
```

4.1.3 What's loaded first? See Paths and inheritance

4.2 Tasks

4.2.1 Shell

Provides tasks for shell commands execution - mostly used in YAML syntax and in Python modules.

4.2.1.1 :sh

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.shell	rkd.standardlib.shell.ShellCommandTask	pip install rkd==SELECT VERSION	

Executes a Bash script. Can be multi-line.

Notice: phrase %RKD% is replaced with an rkd binary name

Example of plain usage:

```
rkd :sh -c "ps aux"
rkd :sh --background -c "some-heavy-task"
```

Example of task alias usage:

```

from rkd.syntax import TaskAliasDeclaration as Task

#
# Example of Makefile-like syntax
#

IMPORTS = []

TASKS = [
    Task(':find-images', [
        ':sh', '-c', 'find ../../ -name \'*.png\''
    ]),

    Task(':build', [':sh', '-c', ''' set -x;
        cd ../../../

        chmod +x setup.py
        ./setup.py build

        ls -la
    '''])
]
```

4.2.1.2 :exec

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.shell	rkd.standardlib.shell.ExecProcessCommand	pip install rkd==SELECT VERSION	

Works identically as **:sh**, but for spawns a single process. Does not allow a multi-line script syntax.

4.2.1.3 Class to import: BaseShellCommandWithArgumentParsingTask

Creates a command that executes bash script and provides argument parsing using Python's argparse. Parsed arguments are registered as ARG_{{argument_name}} eg. –activity-type would be exported as ARG_ACTIVITY_TYPE.

```
IMPORTS += [
    BaseShellCommandWithArgumentParsingTask(
        name=":protest",
        group=":activism",
        description="Take action!",
```

(continues on next page)

(continued from previous page)

```
        arguments_definition=lambda argparse: (
            argparse.add_argument('--activity-type', '-t', help='Select an activity␣
↪type')
        ),
        command='''
            echo "Let's act! Let's ${ARG_ACTIVITY_TYPE}!"
        '''
    )
]
```

4.2.2 Technical/Core

4.2.2.1 :init

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.InitTask	pip install rkd== SELECT VERSION	

This task runs ALWAYS. :init implements a possibility to inherit global settings to other tasks

4.2.2.2 :tasks

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.TasksListingTask	pip install rkd== SELECT VERSION	

Lists all tasks that are loaded by all chained makefile.py configurations.

Environment variables:

- RKD_WHITELIST_GROUPS: (Optional) Comma separated list of groups to only show on the list
- RKD_ALIAS_GROUPS: (Optional) Comma separated list of groups aliases eg. ":international-workers-association->:iwa,:anarchist-federation->:fa"

4.2.2.3 :version

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.VersionTask	pip install rkd== SELECT VERSION	

Shows version of RKD and lists versions of all loaded tasks, even those that are provided not by RiotKit. The version strings are taken from Python modules as RKD strongly rely on Python Packaging.

4.2.2.4 CallableTask

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.CallableTask	pip install rkd== SELECT VERSION	

This is actually not a task to use directly, it is a template of a task to implement yourself. It's kind of a shortcut to create a task by defining a simple method as a callback.

```
import os
from rkd.syntax import TaskDeclaration
from rkd.standardlib import CallableTask
from rkd.contract import ExecutionContext


def union_method(context: ExecutionContext) -> bool:
    os.system('xdg-open https://iwa-ait.org')
    return True


IMPORTS = [
    TaskDeclaration(CallableTask(':create-union', union_method))
]

TASKS = []
```

4.2.2.5 :rkd:create-structure

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.CreateStructureTask	pip install rkd== SELECT VERSION	

Creates a template structure used by RKD in current directory.

4.2.2.6 :file:line-in-file

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.LineInFileTask	pip install rkd== SELECT VERSION	

Similar to the Ansible's lineinfile, replaces/creates/deletes a line in file.

Example usage:

```
echo "Number: 10" > test.txt

rkd -rl debug :file:line-in-file test.txt --regexp="Number: ([0-9]+)?(.*)" --insert=
↪'Number: $match[0] / new: 10'
```

(continues on next page)

(continued from previous page)

```
cat test.txt

rkd -rl debug :file:line-in-file test.txt --regexp="Number: ([0-9]+)?(.*)" --insert=
↪'Number: $match[0] / new: 6'
cat test.txt

rkd -rl debug :file:line-in-file test.txt --regexp="Number: ([0-9]+)?(.*)" --insert=
↪'Number: 50'
cat test.txt

rkd -rl debug :file:line-in-file test.txt --regexp="Number: ([0-9]+)?(.*)" --insert=
↪'Number: $match[0] / new: 90'
cat test.txt
```

4.2.3 Docker

4.2.3.1 :docker:tag

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.docker	rkd.standardlib.docker.TagImageTask	pip install rkd== SELECT VERSION	

Performs a docker-style tagging of an image that is being released - example: 1.0.1 -> 1.0 -> 1 -> latest

Example of usage:

```
rkd :docker:tag --image=quay.io/riotkit/filerepository:3.0.0-RC1 --propagate -rf debug
```

4.2.3.2 :docker:push

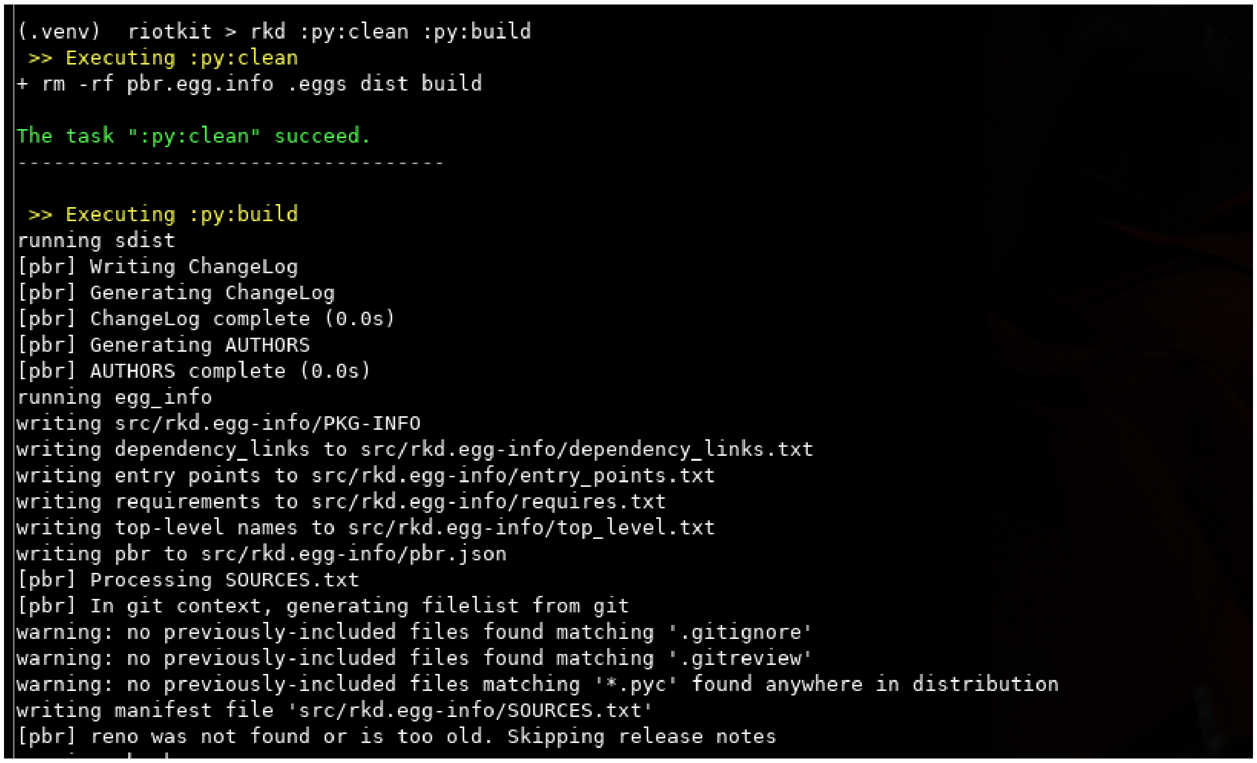
Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.docker	rkd.standardlib.docker.PushTask	pip install rkd== SELECT VERSION	

Does same thing and taking same arguments as **:docker:tag**, one difference - pushing already created tasks.

4.2.4 Python

This package was extracted from standardlib to rkd_python, but is maintained together with RKD as part of RKD core.

Set of Python-related tasks for building, testing and publishing Python packages.



```
(.venv)  riotkit > rkd :py:clean :py:build
 >> Executing :py:clean
+ rm -rf pbr.egg.info .eggs dist build

The task ":py:clean" succeed.
-----------------------------------

 >> Executing :py:build
running sdist
[pbr] Writing ChangeLog
[pbr] Generating ChangeLog
[pbr] ChangeLog complete (0.0s)
[pbr] Generating AUTHORS
[pbr] AUTHORS complete (0.0s)
running egg_info
writing src/rkd.egg-info/PKG-INFO
writing dependency_links to src/rkd.egg-info/dependency_links.txt
writing entry points to src/rkd.egg-info/entry_points.txt
writing requirements to src/rkd.egg-info/requires.txt
writing top-level names to src/rkd.egg-info/top_level.txt
writing pbr to src/rkd.egg-info/pbr.json
[pbr] Processing SOURCES.txt
[pbr] In git context, generating filelist from git
warning: no previously-included files found matching '.gitignore'
warning: no previously-included files found matching '.gitreview'
warning: no previously-included files matching '*.pyc' found anywhere in distribution
writing manifest file 'src/rkd.egg-info/SOURCES.txt'
[pbr] reno was not found or is too old. Skipping release notes
```

4.2.4.1 :py:publish

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.PublishTask	pip install rkd_python== SELECT VERSION	

Publish a package to the PyPI.

Example of usage:

```
rkd :py:publish --username=__token__ --password=.... --skip-existing --test
```

4.2.4.2 :py:build

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.BuildTask	pip install rkd_python== SELECT VERSION	

Runs a build through setuptools.

4.2.4.3 :py:install

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.InstallTask	pip install rkd_python== SELECT VERSION	

Installs the project as Python package using setuptools. Calls ./setup.py install.

4.2.4.4 :py:clean

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.CleanTask	pip install rkd_python== SELECT VERSION	

Removes all files related to building the application.

4.2.4.5 :py:unittest

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.UnitTestTask	pip install rkd_python== SELECT VERSION	

Runs Python's built'in unittest module to execute unit tests.

Examples:

```
rkd :py:unittest
rkd :py:unittest -p some_test
rkd :py:unittest --tests-dir=../test
```

4.2.5 ENV

Manipulates the environment variables stored in a .env file

RKD is always loading an .env file on startup, those tasks in this package allows to manage variables stored in .env file in the scope of a project.

4.2.5.1 :env:get

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.env	rkd.standardlib.env.SetEnvTask	pip install rkd== SELECT VERSION	

Example of usage:

```
rkd :env:get --name COMPOSE_PROJECT_NAME
```

4.2.5.2 :env:set

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.env	rkd.standardlib.env.GetEnvTask	pip install rkd== SELECT VERSION	

Example of usage:

```
rkd :env:set --name COMPOSE_PROJECT_NAME --value hello
rkd :env:set --name COMPOSE_PROJECT_NAME --ask
rkd :env:set --name COMPOSE_PROJECT_NAME --ask --ask-text="Please enter your name:"
```

4.2.6 JINJA

Renders JINJA2 files, and whole directories of files. Allows to render by pattern.

All includes and extends are by default looking in current working directory path.

4.2.6.1 :j2:render

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.jinja	rkd.standardlib.jinja.FileRendererTask	pip install rkd== SELECT VERSION	

Renders a single file from JINJA2.

Example of usage:

```
rkd :j2:render -s SOURCE-FILE.yaml.j2 -o OUTPUT-FILE.yaml
```

4.2.6.2 :j2:directory-to-directory

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.jinja	rkd.standardlib.jinja.FileRendererTask	pip install rkd== SELECT VERSION	

Renders all files recursively in given directory to other directory. Can remove source files after rendering them to the output files.

Pattern is a regexp pattern that matches whole path, not only file name

Example usage:

```
rkd :j2:directory-to-directory \
    --source="/some/path/templates" \
    --target="/some/path/rendered" \
    --delete-source-files \
    --pattern="(.*).j2"
```

4.3 Usage

4.3.1 Importing tasks

Tasks can be defined as installable Python's packages that you can import in your Makefile

Please note:

- To import a group, the package you try to import need to hvve a defined **imports()** method inside of the package.
- The imported group does not need to import automatically dependend tasks (but it can, it is recommended), you need to read into the docs of specific package if it does so

4.3.1.1 1) Install a package

RKD defines dependencies using Python standards.

Example: Given we want to import tasks from package "rkt_armutils".

```
echo "rkt_armutils==3.0" >> requirements.txt
pip install -r requirements.txt
```

Good practices:

- Use fixed versions eg. 3.0 or even 3.0.0 and upgrade only intentionally to reduce your work on fixing bugs

4.3.1.2 2) In YAML syntax

Example: Given we want to import task "InjectQEMUBinaryIntoContainerTask", or we want to import whole "rkt_armutils.docker" group

```
imports:
    # Import whole package, if the package defines a group import (method imports())
    - rkt_armutils.docker

    # Or import single task
    - rkt_armutils.docker.InjectQEMUBinaryIntoContainerTask
```

4.3.1.3 2) In Python syntax

Example: Given we want to import task "InjectQEMUBinaryIntoContainerTask", or we want to import whole "rkt_armutils.docker" group

```
from rkd.syntax import TaskDeclaration
from rkt_armutils.docker import InjectQEMUBinaryIntoContainerTask

# ... (use "+" operator to append, remove "+" if you didn't define any import yet)
IMPORTS += [TaskDeclaration(InjectQEMUBinaryIntoContainerTask)]
```

4.3.2 Troubleshooting

1. Output is corrupted or there is no output from a shell command executed inside of a task

The output capturing is under testing. The Python's subprocess module is skipping "sys.stdout" and "sys.stderr" by writing directly to /dev/stdout and /dev/stderr, which makes output capturing difficult.

Run rkd in compat mode to turn off output capturing from shell commands:

```
RKD_COMPAT_SUBPROCESS=true rkd :some-task-here
```

4.3.3 Loading priority

4.3.3.1 Environment variables loading order from .env and from .rkd

Legend: Top is most important, the variables loaded on higher level are not overridden by lower level

1. Operating system environment
2. Current working directory .env file
3. .env files from directories defined in RKD_PATH

4.3.3.2 Environment variables loading order in YAML syntax

Legend: Top - is most important

1. Operating system environment
2. .env file
3. Per-task "environment" section
4. Per-task "env_file" imports
5. Global "environment" section
6. Global "env_file" imports

4.3.3.3 Order of loading of makefile files in same .rkd directory

Legend: Lower has higher priority (next is appending changes to previous)

1. *.py
2. *.yaml
3. *.yml

4.3.3.4 Paths and inheritance

RKD by default search for .rkd directory in current execution directory - *./.rkd*.

The search order is following (from lower to higher load priority):

1. RKD's internals (we provide a standard tasks like *:tasks*, *:init*, *:sh*, *:exec* and more)
2. */usr/lib/rkd*
3. User's home *~/.rkd*
4. Current directory *./.rkd*
5. *RKD_PATH*

Custom path defined via environment variable

RKD_PATH allows to define multiple paths that would be considered in priority.

```
export RKD_PATH="/some/path:/some/other/path:/home/user/riotkit/.rkd-second"
```

How the makefiles are loaded?

Each makefile is loaded in order, next makefile can override tasks of previous. That's why we at first load internals, then your tasks.

4.3.3.5 Tasks execution

Tasks are executed one-by-one as they are specified in commandline or in TaskAlias declaration (commandline arguments).

```
rkd :task-1 :task-2 :task-3
```

1. task-1
2. task-2
3. task-3

A –keep-going can be specified after given task eg. :task-2 –keep-going, to ignore a single task failure and in consequence allow to go to the next task regardless of result.

4.3.4 Tasks development

RKD has two approaches to define a task. The first one is simpler - in makefile in YAML or in Python. The second one is a set of tasks as a Python package.

4.3.4.1 Creating simple tasks in YAML syntax

Example 1:

```
version: org.riotkit.rkd/0.3
imports:
  - rkd.standardlib.docker.TagImageTask

tasks:
  # see this task in "rkd :tasks"
```

(continues on next page)

(continued from previous page)

```
  # run with "rkd :examples:bash-test"
  :examples:bash-test:
      description: Execute an example command in bash - show only python related tasks
      steps: |
             echo "RKD_DEPTH: ${RKD_DEPTH} # >= 2 means we are running rkd-in-rkd"
             echo "RKD_PATH: ${RKD_PATH}"
             rkd --silent :tasks | grep ":py"

  # try "rkd :examples:arguments-test --text=Hello --test-boolean"
  :examples:arguments-test:
      description: Show example usage of arguments in Bash
      arguments:
          "--text":
              help: "Adds text message"
              required: True
          "--test-boolean":
              help: "Example of a boolean flag"
              action: store_true # or store_false
      steps:
        - |
          #!bash
          echo " ==> In Bash"
          echo " Text: ${ARG_TEXT}"
          echo " Boolean test: ${ARG_TEST_BOOLEAN}"
        - |
          #!python
          print(' ==> In Python')
          print(' Text: %s ' % ctx.args['text'])
          print(' Text: %s ' % str(ctx.args['test_boolean']))
          return True

  # run with "rkd :examples:list-standardlib-modules"
  :examples:list-standardlib-modules:
      description: List all modules in the standardlib
      steps:
        - |
          #!python
          ctx: ExecutionContext
          this: TaskInterface

          import os

          print('Hello world')
          print(os)
          print(ctx)
          print(this)

          return True
```

Example 2:

```
version: org.riotkit.rkd/0.3

environment:
    GLOBALLY_DEFINED: "16 May 1966, seamen across the UK walked out on a nationwide␣
↪strike for the first time in half a century. Holding solid for seven weeks, they␣
↪won a reduction in working hours from 56 to 48 per week "
```

(continues on next page)

(continued from previous page)

```
env_files:
    - env/global.env

tasks:
    :hello:
        description: |
            #1 line: 11 June 1888 Bartolomeo Vanzetti, Italian-American anarchist who␣
↪was framed & executed alongside Nicola Sacco, was born.
            #2 line: This is his short autobiography:
            #3 line: https://libcom.org/library/story-proletarian-life

        environment:
            INLINE_PER_TASK: "17 May 1972 10,000 schoolchildren in the UK walked out␣
↪on strike in protest against corporal punishment. Within two years, London state␣
↪schools banned corporal punishment. The rest of the country followed in 1987."
        env_files: ['env/per-task.env']
        steps: |
            echo " >> ENVIRONMENT VARIABLES DEMO"
            echo "Inline defined in this task: ${INLINE_PER_TASK}\n\n"
            echo "Inline defined globally: ${GLOBALLY_DEFINED}\n\n"
            echo "Included globally - global.env: ${TEXT_FROM_GLOBAL_ENV}\n\n"
            echo "Included in task - per-task.env: ${TEXT_PER_TASK_FROM_FILE}\n\n"
```

Explanation of examples:

1. "arguments" is an optional dict of arguments, key is the argument name, subkeys are passed directly to argparse
2. "steps" is a mandatory list or text with step definition in Bash or Python language
3. "description" is an optional text field that puts a description visible in ":tasks" task
4. "environment" is a dict of environment variables that can be defined
5. "env_files" is a list of paths to .env files that should be included
6. "imports" imports a Python package that contains tasks to be used in the makefile and in shell usage

4.3.4.2 Developing a Python package

Each task should implement methods of **rkd.contract.TaskInterface** interface, that's the basic rule.

Following example task could be imported with path **rkd.standardlib.ShellCommandTask**, in your own task you would have a different package name instead of **rkd.standardlib**.

Example task from RKD standardlib:

```python
class ShellCommandTask(TaskInterface):
    """Executes shell scripts"""

    def get_name(self) -> str:
        return ':sh'

    def get_group_name(self) -> str:
        return ''

    def configure_argparse(self, parser: ArgumentParser):
        parser.add_argument('--cmd', '-c', help='Shell command', required=True)
```

(continues on next page)

(continued from previous page)

```
    def execute(self, context: ExecutionContext) -> bool:
        # self.sh() and self.io() are part of TaskUtilities via TaskInterface

        try:
            self.sh(context.args['cmd'], capture=False)
        except CalledProcessError as e:
            self.io().error_msg(str(e))
            return False

        return True
```

Explanation of example:

1. The docstring in Python class is what will be shown in **:tasks as description**. You can also define your description by implementing **def get_description() -> str**
2. Name and group name defines a full name eg. :your-project:build
3. **def configure_argparse()** allows to inject arguments, and –help description for a task - it's a standard Python's argparse object to use
4. **def execute()** provides a context of execution, please read *Tasks API* chapter about it. In short words you can get commandline arguments, environment variables there.
5. **self.io()** is providing input-output interaction, please use it instead of print, please read *Tasks API* chapter about it.

4.3.4.3 Please check Tasks API for interfaces description

4.3.5 Global environment variables

Global switches designed to customize RKD per project. Put environment variables into your **.env** file, so you will no have to prepend them in the commandline every time.

Read also about *Environment variables loading order from .env and from .rkd*

4.3.5.1 RKD_WHITELIST_GROUPS

Allows to show only selected groups in the ":tasks" list. All tasks from hidden groups are still callable.

Examples:

```
RKD_WHITELIST_GROUPS=:rkd, rkd :tasks
RKD_WHITELIST_GROUPS=:rkd rkd :tasks
```

4.3.5.2 RKD_ALIAS_GROUPS

Alias group names, so it can be shorter, or even group names could be not typed at all.

Notice: :tasks will rename a group with a first defined alias for this group

Examples:

```
RKD_ALIAS_GROUPS=":rkd->:r" rkd :tasks :r:create-structure
RKD_ALIAS_GROUPS=":rkd->" rkd :tasks :create-structure
```

4.3.5.3 RKD_UI

Allows to toggle (true/false) the UI - messages like "Executing task X" or "Task finished", leaving only tasks stdout, stderr and logs.

4.3.5.4 RKD_AUDIT_SESSION_LOG

Logs output of each executed task, when set to "true".

Example structure of logs:

```
# ls .rkd/logs/2020-06-11/11\:06\:02.068556/
task-1-init.log  task-2-harbor_service_list.log
```

4.3.6 Custom distribution

RiotKit Do can be used as a transparent framework for writing tasks for various usage, especially for specialized usage. To simplify usage for end-user RKD allows to create a custom distribution.

Custom distribution allows to:

- Define custom 'binary' name eg. "harbor" instead of "rkd"
- Hide unnecessary tasks in custom 'binary' (filter by groups - whitelist)
- Make shortcuts to tasks: Skip writing group name, make a group name to be appended by default

4.3.6.1 Example

```
import os
from rkd import main as rkd_main

def env_or_default(env_name: str, default: str):
    return os.environ[env_name] if env_name in os.environ else default

def main():
    os.environ['RKD_WHITELIST_GROUPS'] = env_or_default('RKD_WHITELIST_GROUPS', ':env,
↪:harbor,')
    os.environ['RKD_ALIAS_GROUPS'] = env_or_default('RKD_ALIAS_GROUPS', '->:harbor')
    os.environ['RKD_UI'] = env_or_default('RKD_UI', 'false')
    rkd_main()


if __name__ == '__main__':
    main()
```

```
$ harbor :tasks
[global]
:sh                                                 # Executes shell scripts
:exec                                               # Spawns a shell process
:init                                               # :init task is executing ALWAYS.␣
↪That's a technical, core task.
:tasks                                              # Lists all enabled tasks
:version                                            # Shows version of RKD and of all␣
↪loaded tasks
```

(continues on next page)

(continued from previous page)

```
[harbor]
:compose:ps                                       # List all containers
:start                                            # Create and start containers
:stop                                             # Stop running containers
:remove                                           # Forcibly stop running containers␣
↪and remove (keeps volumes)
:service:list                                     # Lists all defined containers in␣
↪YAML files (can be limited by --profile selector)
:service:up                                       # Starts a single service
:service:down                                     # Brings down the service without␣
↪deleting the container
:service:rm                                       # Stops and removes a container and␣
↪it's images
:pull                                             # Pull images specified in␣
↪containers definitions
:restart                                          # Restart running containers
:config:list                                      # Gets environment variable value
:config:enable                                    # Enable a configuration file - YAML
:config:disable                                   # Disable a configuration file -␣
↪YAML
:prod:gateway:reload                              # Reload gateway, regenerate␣
↪missing SSL certificates
:prod:gateway:ssl:status                          # Show status of SSL certificates
:prod:gateway:ssl:regenerate                      # Regenerate all certificates with␣
↪force
:prod:maintenance:on                              # Turn on the maintenance mode
:prod:maintenance:off                             # Turn on the maintenance mode
:git:apps:update                                  # Fetch a git repository from the␣
↪remote
:git:apps:update-all                              # List GIT repositories
:git:apps:set-permissions                         # Make sure that the application␣
↪would be able to write to allowed directories (eg. upload directories)
:git:apps:list                                    # List GIT repositories

[env]
:env:get                                          # Gets environment variable value
:env:set                                          # Sets environment variable in the .
↪env file


Use --help to see task environment variables and switches, eg. rkd :sh --help, rkd --
↪help
```

Notices for above example:

- No need to type eg. :harbor:config:list - just :config:list (RKD_ALIAS_GROUPS used)
- No "rkd" group is displayed (RKD_WHITELIST_GROUPS used)
- There is no information about task name (RKD_UI used)

4.3.6.2 Read more in Global environment variables

4.3.7 Tasks API

4.3.7.1 Each task must implement a TaskInterface

class `rkd.contract.`**`TaskInterface`**

`configure_argparse`(*parser: argparse.ArgumentParser*)
Allows a task to configure ArgumentParser (argparse)

`copy_internal_dependencies`(*task*)
Allows to execute a task-in-task, by copying dependent services from one task to other task

`exec`(*cmd: str*, *capture: bool = False*, *background: bool = False*) → Optional[str]
Starts a process in shell. Throws exception on error. To capture output set capture=True

NOTICE: Use instead of subprocess. Raw subprocess is less supported and output from raw subprocess
may be not catch properly into the logs

`execute`(*context: rkd.contract.ExecutionContext*) → bool
Executes a task. True/False should be returned as return

`format_task_name`(*name: str*) → str
Allows to add a fancy formatting to the task name, when the task is displayed eg. on the :tasks list

`get_declared_envs`() → Dict[str, str]
Dictionary of allowed envs to override: KEY -> DEFAULT VALUE

`get_full_name`()
Returns task full name, including group name

`get_group_name`() → str
Group name where the task belongs eg. ":publishing", can be empty.

`get_name`() → str
Task name eg. ":sh"

`io`() → rkd.inputoutput.IO
Gives access to Input/Output object

`rkd`(*args: list*, *verbose: bool = False*, *capture: bool = False*) → str
Spawns an RKD subprocess

NOTICE: Use instead of subprocess. Raw subprocess is less supported and output from raw subprocess
may be not catch properly into the logs

`sh`(*cmd: str*, *capture: bool = False*, *verbose: bool = False*, *strict: bool = True*, *env: dict = None*) → Optional[str]
Executes a shell script in bash. Throws exception on error. To capture output set capture=True

NOTICE: Use instead of subprocess. Raw subprocess is less supported and output from raw subprocess
may be not catch properly into the logs

`silent_sh`(*cmd: str*, *verbose: bool = False*, *strict: bool = True*, *env: dict = None*) → bool
sh() shortcut that catches errors and displays using IO().error_msg()

NOTICE: Use instead of subprocess. Raw subprocess is less supported and output from raw subprocess
may be not catch properly into the logs

static table(*header: list*, *body: list*, *tablefmt: str = 'simple'*, *floatfmt: str = 'g'*, *numalign: str = 'decimal'*, *stralign: str = 'left'*, *missingval: str = ''*, *showindex: str = 'default'*, *disable_numparse: bool = False*, *colalign: str = None*)

Renders a table

Parameters

- **header** –
- **body** –
- **tablefmt** –
- **floatfmt** –
- **numalign** –
- **stralign** –
- **missingval** –
- **showindex** –
- **disable_numparse** –
- **colalign** –

Returns Formatted table as string

4.3.7.2 Execution context provides parsed shell arguments and environment variables

class rkd.contract.**ExecutionContext**(*declaration: rkd.contract.TaskDeclarationInterface*, *parent: Optional[rkd.contract.GroupDeclarationInterface] = None*, *args: Dict[str, str] = {}*, *env: Dict[str, str] = {}*)

Defines which objects could be accessed by Task. It's a scope of a single task execution.

get_arg(*name: str*) → Optional[str]

Get argument or option

Usage: ctx.get_arg('–name') # for options ctx.get_arg('name') # for arguments

Raises KeyError when argument/option was not defined

Returns Actual value or default value

get_arg_or_env(*name: str*) → Optional[str]

Provides value of user input

Usage: get_arg_or_env('–file-path') resolves into FILE_PATH env variable, and –file-path switch (file_path in argparse)

Behavior: When user provided explicitly switch eg. –history-id, then it's value will be taken in priority. If switch –history-id was not used, but user provided HISTORY_ID environment variable, then it will be considered.

If no switch provided and no environment variable provided, but a switch has default value - it would be returned. If no switch provided and no environment variable provided, the switch does not have default, but environment variable has a default value defined, it would be returned.

Raises `MissingInputException` – When no switch and no environment variable was provided, then an exception is thrown.

get_env (*name: str*, *error_on_not_used: bool = False*)
Get environment variable value

4.3.7.3 Interaction with input and output

class rkd.inputoutput.**IO**
Interacting with input and output - stdout/stderr/stdin, logging

capture_descriptors (*target_files: List[str] = None*, *stream=None*, *enable_standard_out: bool = True*)
Capture stdout and stderr from a block of code - use with 'with'

critical (*text*)
Logger: critical

debug (*text*)
Logger: debug

err (*text*)
Standard error

errln (*text*)
Standard error + newline

error (*text*)
Logger: error

error_msg (*text*)
Error message (optional output)

h1 (*text*)
Heading #1 (optional output)

h2 (*text*)
Heading #2 (optional output)

h3 (*text*)
Heading #3 (optional output)

h4 (*text*)
Heading #3 (optional output)

info (*text*)
Logger: info

info_msg (*text*)
Informational message (optional output)

is_silent () → bool
Is output silent? In silent mode OPTIONAL MESSAGES are not shown

opt_out (*text*)
Optional output - fancy output skipped in –silent mode

opt_outln (*text*)
Optional output - fancy output skipped in –silent mode + newline

out (*text*)
Standard output

outln (*text*)
Standard output + newline

print_group(*text*)
Prints a colored text inside brackets [text] (optional output)

print_line()
Prints a newline

print_opt_line()
Prints a newline (optional output)

print_separator()
Prints a text separator (optional output)

success_msg(*text*)
Success message (optional output)

warn(*text*)
Logger: warn

Index