

# The CTA Operator tools public repository

In which we discuss packaging and tagging

Richard Bachmann

# Outline

Goals

Repository structure

Packaging with pip

Tagging and releasing

Using the packages

# Goals

- Provide operator tools “as is” to the CTA community
  - Needs: Mechanism of installing and upgrading
  - Needs: Config management
- Use the same tools ourselves
- Host example monitoring configs

# Repository structure

- <https://gitlab.cern.ch/cta/cta-operations>

## Project:

```
.
ci_helpers <-- Misc. CI utilities
cta-ops-config.yaml <-- Reference config file
monitoring
  grafana
    dashboards <-- Dashboard json and previews
    td-agent <-- Fluentd config files
LICENSE
README.md
requirements.txt <-- Full install requirements list
rpm <-- RPM package(s) for general setup
tools
  pip <-- Individual tools, written in Python
    ctutils <-- Misc. utilities
    tapeadmin <-- Tape interaction utilities
    cta-ops-repack-automation <-- Repack automation tools
```

## Pip package:

```
LICENSE
Makefile
pyproject.toml
README.md
src
  atresys
    email_templates
    cta_ops_repack_0_scan.py
    ...
    __init__.py
```

# Building with pip

## Specifications

- PEP 517 - A build-system independent format for source trees
  - <https://peps.python.org/pep-0517/>
- PEP 518 – Specifying Minimum Build System Requirements for Python Projects
  - <https://peps.python.org/pep-0518/>
- PEP 503 – Simple Repository API
  - <https://peps.python.org/pep-0503/>
- Python Packaging Authority (PyPA) tutorials and specs
  - <https://packaging.python.org/en/latest/#>

# Build tools

## Build frontend

*A build frontend is a tool that users might run that takes arbitrary source trees or source distributions and builds wheels from them.*

- We use 'build'
  - wrapped by `make`

## Build backend

*The actual building is done by each source tree's build backend.*

- We use 'setuptools'

# Package metadata

## All in the toml file

- `setup.cfg`,
- `setup.py`
- `pyproject.toml`
- [https://setuptools.pypa.io/en/latest/userguide/pyproject\\_config.html](https://setuptools.pypa.io/en/latest/userguide/pyproject_config.html)

```
[build-system]
requires = ["setuptools>=61.2", "setuptools_scm>=6.2"]
build-backend = "setuptools.build_meta"

[project]
name = "atresys"
authors = [
    {name = "CERN", email = "tape-operations@cern.ch"},
]
description = "Tools for automating tape repack workflows"
readme = "README.md"
requires-python = ">=3.6"
license = {file = "LICENSE"}
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: GNU General Public License V3 (GPLV3)",
    "Operating System :: POSIX :: Linux"
]
keywords = ["CTA", "tape", "CERN"]

dynamic = ["version"]
dependencies = [
    "ctautils",
    "tapeadmin",
    "tabulate"
]

[project.urls]
repository = "https://gitlab.cern.ch/cta/cta-operations/"
documentation = "https://gitlab.cern.ch/cta/cta-operations/-/wikis/tools/ATRESYS---Automated-Tape-Repack-Manager"

[project.scripts]
cta-ops-repack-manager = "atresys.cta_ops_repack_manager.main"
...
```

# Tagging convention

## Present internal repo situation

- 0.4-123
  - '0.4' - unused
  - '-123' - incremented when new release is created

## Proposal (but we could use anything else really)

- 5.8.6.1
  - '5.8.6' - **min** CTA version needed to run
    - Update if backwards incompatible cta-admin changes
  - '.1' - Incremented when new ops release is made, set to 0 when CTA min version changes



# Versioning with the build system

## Setuptools-scm

- Allows us to build with dynamic version from CI tags
  - <https://pypi.org/project/setuptools-scm/>
- Does intermediate versions based on tag

*In the standard configuration setuptools-scm takes a look at three things:*

- *latest tag (with a version number)*
- *the distance to this tag (e.g. number of revisions since latest tag)*
- *workdir state (e.g. uncommitted changes since latest tag)*

1. Last tag: 5.8.6.1
2. Code then `git commit`
3. New intermediate version: 5.8.6.**2**.dev1
4. Publish intermediate on public repo for testing, *don't tag*
5. When happy, tag and publish 5.8.6.2

# Using a tag

## Q: Which version should I use?

Assuming the proposed scheme:

- **CERN:** Most recent tagged release or specific dev release
- **Elsewhere:** Closest non-dev release with version  $\leq$  CTA version




# Gitlab CI

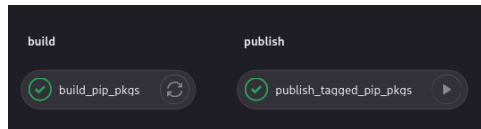
- Uses EOSWeb space to publish
- Job for publishing to Gitlab repo available, publishing to PyPI possible

## Pipeline

1. Build (simply runs `make`)
2. Publish (manual)
  - Publish dev commits for testing at CERN
  - Publish tags as 'public release'

## Index of /

<a href="#">Name</a>	<a href="#">Last_modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">cta-4/</a>	2022-10-04 17:13	-	
 <a href="#">cta-5/</a>	2022-10-04 17:10	-	
 <a href="#">cta-operations/</a>	2023-03-17 14:22	-	



# Using the packages

## Pip as first-class citizen:

Available using index url:

- `https://cta-public-repo.web.cern.ch/cta-operations/pip/simple/`

For easy install we provide:

- requirements.txt with external dependency versionlock

```
python3 -m pip install --extra-index-url https://cta-public-repo.web.cern.ch/cta-operations/pip/simple/  
--requirement requirements.txt
```

- Users can also define it as a repo in their pip.conf file

# Rpm wrapper

Single 'just install everything' rpm. Created when we tag.

1. Install 'tape-local' user
2. Set up venv in \$PATH
  - Avoid system-level pip pkg interference
  - Avoid custom/hacky *opt...* path
3. `pip install --r requirements.txt`
4. TODO: keytab setup?

# In practice

## CERN - make a tool public

1. Move tool source code of tool to new Gitlab repo (new pip pkg)
2. Add tool to 'ctaops-lib' requirements.txt
  - Treat like external dependency
  - Versionlock
  - Separate ops and dev release timings

Packages are copied to our internal mirror, fetched by hosts from there. Can use public repo directly if we want to.

## Elsewhere

- Simple requirements.txt for 'pip install everything'
  - Or install by hand with pip
- Use RPM
- Use container (creation for summer student)



# Bonus slide: Why not 0.1.2-style tags?

- Implies semantic versioning
- Indicates that we differentiate between major/minor/patch releases, which we don't