tyrannosaurus

Release 0.11.0

Douglas Myers-Turnbull

Jul 10, 2023

CONTENTS

1 Making a new project 1

2 Building and testing your project 3

3 To-do reference for new projects 5

4 Metadata synchronization 9

5 Working with Anaconda 11

6 Testing database-connected code 13

7 Reference of files 15

8 Making a new template 19

9 API Reference 21

10 What's wrong with pip or anaconda? 33

11 List of integrations 35

Python Module Index 37

Index 39

CHAPTER

ONE

MAKING A NEW PROJECT

You can use Tyrannosaurus to create a new project. Run: `tyrannosaurus new myproject --track`.

1.1 Basic usage of `tyrannosaurus new`

Internally, it clones the Tyrannosaurus repo, checks out the correct version, and fills your new repo with the proper files. The `--track` flag causes it to run `git init` and track the `main` branch of the repo at `https://github.com/{user}/myproject.git`. The user will be guessed from your git config, but it can be set with `--user`. It's fine to pass the name of a GitHub org, too.

Tip: You need to create the repo on GitHub before running `new`. If you didn't do this, delete the `.git` dir, run `git init --initial-branch=main`, and run `git remote add origin https://github.com/{user}/{myproject}.git`. (You can also just replace the bad `.git` with the correct one from your GitHub repo.)

But you can pass those in with `--authors` (comma-separated), `--version`, `--status`, and `--keywords`. You can choose a different license using `--license`. Choices are Apache 2, CC0, CC-BY, CC-BY-NC, GPL 3, LGPL, and MIT. See `tyrannosaurus new --help` for more info.

Note: GitHub made *main* the default branch name for new repositories in October 2020. The provided workflows should work with either *main* or *master*. You'll want `git push origin main`, **not** `git push origin master`, unless you renamed the branch. I recommend keeping *main* as the default branch.

1.2 Project metadata in pyproject.toml to modify

Modify your project as needed after running `tyrannosaurus new`. Most of the metadata you'll want to modify is contained in `pyproject.toml`. There are a bunch of comments with `TODO:` in this file. Chances are, you will only need to modify those. In particular, two sections:

- *Poetry metadata* (`[tool.poetry]` and `[tool.poetry.urls]`)
- *Poetry build & dependencies* (`[tool.poetry.dev-dependencies]`, etc.)

Tip: **Support for Python < 3.8:** If you need to support Python 3.7 and below, add `importlib_metadata` to `pyproject.toml` and `docs/requirements.txt`. Then change `importlib.metadata` in `__init__.py` to `importlib_metadata`.

1.3 Readme file, issue labels, etc.

You may want to modify `.github/settings.json` (or `.github/labels.json`); when you commit, your full settings (or just GitHub labels) will be replaced with these. Chances are you'll also want to modify the readme :)

> **Caution:** Will replace your GitHub labels each time you commit. Either edit the `.github/labels.json` file or disable by deleting `.github/workflows/labels.yml`.

1.4 Manual steps to configure reports

> **Hint:** Also see the new project guide. It has a more complete list of steps; this doc contains more discussion and explanation.

You will need to set a few GitHub tokens and set up a few external services manually.

Coverage reports will be sent to Coveralls and/or CodeCov. Code quality analysis will also be performed using CodeClimate and GitHub's CodeQL. The default readme file shows shields for Coveralls, CodeCov, and Code Climate; these always reflect what's in your main branch. All four of these are free for open source projects. You probably only want *either* Coveralls or CodeCov because they do the same thing, whereas Code Climate provides maintainability summaries, and CodeQL provides security checks.

These analyses are run whenever you push to the main branch, handled by the workflow file `.github/workflows/commit.yml`. Code Climate tracks your main branch on its own through a GitHub push webhook.

Coveralls and CodeCov will only work if you add their tokens to your GitHub repo secrets. If these tokens aren't found or are invalid, the workflow commands will silently fail. (This is so that you don't have to use both, or even either.) All of these will also need webhooks added, though they currently can do that automatically after you authorize them on GitHub.

Here's what you need to do to set these up:

- Connect CodeClimate and either Coveralls or CodeCov and to your GitHub account and follow their configuration instructions.
- Add `CODECOV_TOKEN` to your GitHub repo secrets, if needed.
- Code Climate assigns a url for your repo. Currently, can see it in Settings→Badges. For example, the badge link for Tyrannosaurus is:

  `https://api.codeclimate.com/v1/badges/5e3b38c9b9c418461dc3/maintainability`.

  Copy that URL to `README.md`.

CHAPTER

TWO

BUILDING AND TESTING YOUR PROJECT

This section assumes you will be using Poetry and Tox. If you don't have Poetry, you should install it. To will be installed as a dependency when you run `poetry install`.

After that, you can always run `tox` to build and run tests. The `tox.ini` runs a sequence of commands to build, synchronize, clean, and test your project; create a new `poetry.lock`; and generate HTML docs. Take a look at the `testenv` section to see the commands. The GitHub workflows `build.yml` and `publish.yml` install Poetry and run Tox, so whatever you add to `tox.ini` will be done in GitHub.

Tip: You can bump major or minor versions of your dependencies using `poetry update`.

CHAPTER

THREE

TO-DO REFERENCE FOR NEW PROJECTS

This doc is linked by Tyrannosaurus after running `tyrannosaurus new`. It may be especially helpful for users new to Python packaging. See the new project guide for usage of `new` and more detailed information about some of these steps.

3.1 To-do list

These are suggested steps for after running `tyrannosaurus new --track`. The `--track` option to the `new` command tracks `usernameororg/project`. Alternatively, copy the files Tyrannosaurus generated into your repo, or use `--track` and just modify the git remote afterward.

Tip: Check `git config user.name`, `git config user.name`, and your GPG keys: `git config --global user.signingkey`

3.1.1 Getting your first commit on GitHub

1. **Create a repo:** Create an *non-initialized* repository under `userororgname/projectname`. (Don't add a readme, etc.) Install Probot Settings: in your repo. 3. **Delete unwanted files:** Delete `.github/labels.json`, `.github/workflows/labels.yml`, `.travis.yml`, and `.azure-pipelines.yml`, assuming you're not using them. You might also not want `codemeta.json`, `CITATION.cff`, `environment.yml`, or `Vagrantfile`. 4. Modify `pyproject.toml` and `README.md`. 5. Add an entry to `CHANGELOG.md` and run `git commit -m "feat:  initial code"` (probably twice).

Tip: If a linter such as black fails on commit, just run git commit again to accept the linted version.

Tip: Always run `tyrannosaurus sync` or `poetry lock` after modifying dependencies.

Tip: A Commitizen pre-commit hook checks commit messages. To enable it, you need to run `pre-commit install --hook-type commit-msg`.If it's too restrictive, modify `[tool.commitizen]` in pyproject.toml or remove the pre-commit-config entry.

3.1.2 Configuring external integrations

External services require additional steps. These are making accounts, installing GitHub apps, adding webhooks, and adding GitHub secrets. See the report config guide for discussion.

1. **PyPi:** On PyPi, create a new project and get a repo-specific token. In your GitHub secrets page (under *Settings*), add `PYPI_TOKEN`.
2. **Docker Hub:** Tell DockerHub to track your repo with source `/v[0-9]+.*/` and tag `{sourceref}`. On your GitHub repo Settings Webhooks your docker hook Edit, check `Releases`.
5. **Readthedocs:** Set up readthedocs to track your repo.
6. **Code quality / coverage (1):** Set up Code Climate, Scrutinizer, and either CodeCov or Coveralls.
7. **Code quality / coverage (2):** Add `CODECOV_TOKEN` to your GitHub secrets, if you're using CodeCov
8. **Slack notifications:** For important success/failure notifications, add `SLACK_WEBHOOK` as a GitHub secret and set `use_slack=true` in `.github/action-options.json`.

Tip: If you choose not to install Probot Settings, check out the suggested branch protection rules in `.github/settings.yml`. Also confirm that Dependabot *alerts* and *security updates* are enabled under *Security & analysis*. Currently, Dependabot updates are only available on a GitHub organization.

3.1.3 Your first GitHub release

Follow these steps to publish to GitHub releases, the GitHub Container Registry, Docker Hub, and PyPi. If you're using the default Azure-pipelines file, you'll also publish to Azure's container registry. If you want to publish to Conda-Forge, follow the anaconda recipe-generation steps after the package is on PyPi to get an initial recipe (`meta.yaml`) file. Your project's `CONTRIBUTING.md` will list the steps to make subsequent releases. (See Tyrannosaurus's own contributing guide.)

3.1.4 Final thoughts

Note the normal way to update the *main* branch: 1. Push to a remote branch or fork and make a pull request against *main* branch. 2. When the pull request tests pass, submit a review on the pull request, and rebase it into *main*.

Review Semantic Versioning, Keep a Changelog, and Conventional Commits. Consider getting a DOI for scientific software. If you're using Jupyter, consider adding [nbstripout](https://github.com/kynan/nbstripout) to your `pre-commit-config.yaml` to avoid ending up with a massive git history. If you want to create a package on Conda-Forge, see the anaconda integration guide.

You may want to add new code quality integrations, like codacy. Consider adding shields for those. Other good tools to consider include github-labeler and Towncrier.

Commitizen can be used to generate a changelog. In my experience, those changelogs are not great because (1) commit messages are too messy, (2) the `feat:`, `fix:`, etc. commit types don't match up with those in keep-a-changelog, (3) it fails completely if one commit message is off, (4) it's hard to modify the style at a later date without completely rewriting the git history or adding a plugin for Commitizen, and (5) Commitizen destroys any extra text you add to your Changelog, such as a "Conventions" section. Instead, I just add to the changelog manually.

Warning: Probot Settings has a privilege escalation issue. Either accept that as a caveat, list `.github/settings.yml` in `.github/CODEOWNERS`, or disable it after your initial push.

Caution: Both `tyrannosaurus sync` and Commitizen's `bump` copy version numbers. They won't always play well together. I recommend not using it. In the future, you may be able to point `tool.tyrannosaurus.sources.version` to `tool.commitizen.version` (leaving `tool.commitizen.version_files` empty).

3.2 Reference of commands

These commands might be useful:

- `tyrannosaurus sync` to sync metadata and nothing else
- `tyrannosaurus clean --aggressive` to remove lots of temp files
- `tox` to build, test, build docs, and run some static analyses
- `poetry update` to find updated dependency versions (major or minor)
- `tyrannosaurus recipe` to generate a Conda recipe

These commands are run automatically via either Tox or a GitHub action, but you can run them locally too:

- `poetry install` to install and nothing more
- `poetry build` to build wheels and sdists
- `poetry publish` to upload to PyPi
- `docker build .` to build a docker image

CHAPTER

FOUR

METADATA SYNCHRONIZATION

4.1 Tyrannosaurus sync

Running Tox calls `tyrannosaurus sync`. This command copies some project metadata from pyproject.toml to other files and between sections. These are the required Python versions, development requirements, and line length. It will also warn you if other information seems inconsistent, such as a license file not matching.

You can configure this behavior in pyproject.toml under `[tool.tyrannosaurus.sources]` and `[tool.tyrannosaurus.targets]`. `targets` specifies what files and directories to update. Filename extensions and some directory names are omitted. The information to update will be decided from the filenames. For example, `init` will include the copyright statement from `tool.tyrannosaurus.sources.copyright`. The sources can be either literal values surrounded in single quotes, or the names of other settings in pyproject.toml. You can use `${today}` to refer to the current date and `${datetime}` for the datetime. `datetime` will be in the format `2020-05-07 20:21`. You can access individual fields as expected, such as `${datetime.hour}}` for `'20'`.

Note: Tyrannosaurus always generates backups before modifying. These are saved in `.tyrannosaurus` but are cleared on the next Tox build.

4.2 List of sync targets

Here are most of the available synchronization targets:

- Copyright, status, and date in `__init__.py`
- Dev dependencies between `tool.poetry.dev-dependencies`, `tool.poetry.extras`, and `tox.ini`
- An `all` optional dependency list with all optional non-dev packages
- Dependencies for building docs in `docs/conf.py` and `docs/requirements.txt`
- Code line length between `black` and `pycodestyle`
- Python version in `pyproject.toml`, `tox.ini`, `.travis.yml`, `black`, and `readthedocs.yml`
- Copyright in `docs/conf.py`
- Poetry version in `Dockerfile`
- Authors and year listed in the license file
- Metadata in `CITATION.cff` and `codemeta.json`
- Dev versions in `.pre-commit-config.yaml`

- `--maintainers` arg for Grayskull in `tox.ini`
- `doc_url`, `dev_url`, and `license_file` in `meta.yaml`
- Most recent version in `CHANGELOG.md` assuming Keep a Changelog

CHAPTER

FIVE

WORKING WITH ANACONDA

Tox uses virtualenv, typically creating a new environment for each build. Unfortunately, Tox does not support Anaconda. However, you can use tox on top of a Conda installation without issue. For example, you might have a *build* environment and run tox inside it:

```
conda create \
  --name build \
  --override-channels \
  --channel conda-forge \
  --force \
  --yes \
  python=3.9
  pip install tox poetry
```

In addition, for several reasons, I strongly recommend using Poetry to manage dependencies instead of Anaconda. (Also, I recommend using Miniconda and conda-forge, as I recommended to a user who broke their conda base.)

5.1 If you need a dependency only available via Conda

Unfortunately, you may occasionally have dependencies that are available through Anaconda but not PyPi. One such offending package is rdkit, and there has been a lot of discussion about this issue. Aside from a handful of scientific packages, this is a rare situation. Likely, the best choice is to extract the dependency into a new project. Your dependency can be built with conda, while your main package can be built and tested in CI independently of it. Integration tests can then live outside of both projects. See chemserve for an example solution.

If you really need the dependency coupled directly to your main code, document the requirement in your readme and add it to your GitHub Actions workflows. [Ensureconda](https://pypi.org/project/ensureconda/) and tox-conda may be helpful.

5.2 Anaconda environment file

You can generate an environment yml file using: `tyrannosaurus env`. It will check for packages on Conda-Forge and move packages not found to the `pip:` section.

5.3 Anaconda recipes

This describes how to generate a Conda recipe and [upload it to Conda-Forge. Your desired version must already be published on PyPi, which happens shortly after you create a GitHub release.

5.3.1 Initial recipe generation

You will only need to do this once:

1. Run `tyrannosaurus recipe`, which will run grayskull.
2. Check over your new recipe in `recipes/projectname/meta.yaml`. The recipe-generation command isn't perfect, so fix errors if there are any.
3. Fork from staged-recipes. Check out that repo in a new branch named the same as your project.
4. Copy your recipe from `recipes/projectname/meta.yaml` into that fork (same path), along with your license file.
5. Make a pull request. Tests in Azure will be run, and a maintainer will merge your request.

Tip: On Windows, you may need to run `conda install m2-patch` first.

If everything goes well, your package will be on Conda-Forge soon after.

5.3.2 Keeping the package up-to-date

A new feedstock repo will also be created. For example, see Tyrannosaurus's own feedstock. You should receive an email that adds you to Conda-Forge. Accept it to gain write access to this feedstock. The same goes for anyone else listed as a maintainer in the recipe under `recipe-maintainers:`. A little after each new release on PyPi, you will get a pull request on the feedstock that bumps the version. Let the Azure tests pass, check over the diff – especially making sure that any changes to the dependencies are reflected correctly – then merge. That's it: the conda-forge package should be automatically updated.

CHAPTER

SIX

TESTING DATABASE-CONNECTED CODE

Getting GitHub actions and Tox to work well with databases takes a little extra work. This shows how to get MariaDB/MySQL working. The principles should be the same for other databases, like PostgreSQL. This project has a working example of this.

First, include MariaDB as a service by adding this under `jobs` in `.github/workflows/commit.yml` and `.github/workflows/pull.yml`. Note that the `MYSQL_DATABASE: test` does not refer to your database.

```
services:
    mysql:
        image: mariadb:latest
        env:
            MYSQL_ROOT_PASSWORD: root
            MYSQL_DATABASE: test
        ports:
            - 3306:3306
        options: \
            --health-cmd="mysqladmin ping" \
            --health-interval=10s \
            --health-timeout=5s \
            --health-retries=3
```

Then, add a step to test the MariaDB connection. Add it before other steps so the workflow fails early.

```
-
    name: Initialize MariaDB
    run: |
        mysqladmin --host=127.0.0.1 ping
```

Then, add the SQL schema and rows needed for the tests to `tests/resources/testdb.sql`. There's no security relevance here, so we can just use the root throughout.

Warning: Make sure the name of your test database won't ever conflict with a real database. Otherwise, you'll lose your database.

Then, in `tox.ini`, `mysql` to `whitelist_externals`. Then add this to the `commands`. It's likely to be fast, so consider adding it as the first step.

```
mysql -e 'SOURCE tests/resources/testdb.sql;' --host=127.0.0.1 --user=root --
↪password=root
```

Oddly, the `-host=127.0.0.1` is important; "localhost" or leaving it as default won't work. You may also want to execute a `DROP DATABASE` query as the last command, but that may not be needed.

Your SQL file might start with something like:

```
DROP DATABASE IF EXISTS myfakedatabase;
CREATE DATABASE myfakedatabase CHARACTER SET = 'utf8mb4' COLLATE 'utf8mb4_unicode_520_ci
↪';
DROP USER IF EXISTS 'myfakeuser'@'localhost';
CREATE USER 'myfakedatabase'@'localhost' IDENTIFIED BY 'fakeuser123';
GRANT SELECT, INSERT, UPDATE, DELETE ON myfakedatabase.* TO 'myfakeuser'@'localhost';
USE valartest;
```

The SQL is executed using the root user, but your code may expect something different.

CHAPTER

SEVEN

REFERENCE OF FILES

This section explains what the files are and why they exist.

7.1 Files that Tyrannosaurus created

Some of these can be deleted if you are not using those tools.

filename	purpose
.dockerignore	Tells Docker to ignore temp paths
.editorconfig	Tells IDEs how to handle indentation for different file types
.gitignore	Avoids committing temp, build, and other unwanted files
.pre-commit-config.yaml	Forces checks that prevent unnecessary fixup commits
.travis.yml	Remove, assuming you are not using Travis
.scrutinizer.yml	Configuration for scrutinizer build, coverage, and/or code quality analysis
azure-pipelines.yml	Remove, assuming you are not using Azure Pipelines
CHANGELOG.md	Always keep a changelog.
CITATION.cff	Recommends citations (obscure)
codemeta.json	Documents scientific software (obscure)
CONTRIBUTING.md	Policy on how to contribute (recognized by GitHub)
Dockerfile	Tells Docker how to build the code
Vagrantfile	Vagrantfile for consistent dev environments
LICENSE.txt	License file as plain text (recognized by GitHub)
poetry.lock	Generated by Poetry for consistent and fast dependency resolution
pyproject.toml	Core metadata and config file used by a large number of tools
README.md	Obviously needed (recognized by GitHub)
readthedocs.yml	Tells readthedocs how to build the documentation
SECURITY.md	Security policy document. (recognized by GitHub)
tox.ini	Pipeline file for Tox to build and test the code
.github/ISSUE_TEMPLATE	Tells GitHub to suggest templates for new issues
.github/workflows/checklinks.yml	GitHub project settings via Probot
.github/workflows/codeql.yml	Workflow for GitHub CodeQL action
.github/workflows/commit.yml	Builds, tests, and updates code quality reports on pushing to *main*
.github/workflows/labels.yml	Sets GitHub issue labels specified in .github/labels.json on push
.github/workflows/publish.yml	Publishes when a GitHub release is made (does **not** test)
.github/workflows/pull.yml	Tests, but does not update reports, on pull request to *main*
.github/workflows/settings.yml	GitHub project settings via Probot
dependabot.yml	Tells dependabot to run weekly
labels.json	Specifies the GitHub issue labels to create or update

continues on next page

Table 1 – continued from previous page

filename	purpose
PULL_REQUEST_TEMPLATE.md	Requests a reference to an issue for pull requests (used by GitHub)
docs/conf.py	Configuration file for building documentation with Sphinx
docs/index.rst	reStructuredText documentation main page
docs/requirements.txt	Dependencies required by readthedocs
recipes/*/meta.yaml	A Conda-Forge recipe that *tyrannosaurus recipe* can generate
environment.yml	A Conda environment file that *tyrannosaurus env* can generate
tests/__init__.py	Utilities for tests
tests/resources/	Resource files that tests need
tests/**/test_*.py	Test source files
{pkg}/__init__.py	Declares dunder metadata and provides project utilities
{pkg}/resources/	Resource files that the main code needs
{pkg}/cli.py	Command-line interface with Typer
{pkg}/**/*.py	Main code files!

7.2 Temporary files

These are temporary files, which you can delete.

filename	purpose
.tox/	Cache from Tox containing virtual environments (*avoid deleting*)
docs/autoapi/	Generated reStructuredText files from code docstrings (can delete)
docs/html/	Generated HTML files for all documentation (can delete)
dist/	Poetry-generated sdists and wheels files (can delete)
.tyrannosaurus/	Temporary backup and trashed files from Tyrannosaurus (can delete)
.coverage	Generated coverage reports (can delete)
eggs/	Obsolete generated egg files (auto-deleted)
{pkg}.egg-info	Obsolete generated egg files (auto-deleted)
**/.pytest_cache/	Cache files from pytest (auto-deleted)

7.3 Non-included files

These may be useful for some projects, or are alternatives to those used.

filename	purpose

AUTHORS.md Perfectly fine to include if there are many authors (duplicates info) BACKERS.md Perfectly fine to include CODE_OF_CONDUCT.md Perfectly fine to include codecov.yml May be useful to include when using codecov HISTORY.md Perfectly valid, but CHANGELOG.md seems more clear Makefile Could be used to automate additional development tasks

7.4 Bad/obsolete files

These are files that you specifically should not use.

filename	purpose
dev-requirements.txt	Obsolete, and has serious problems
LICENSE	No-extension filenames cause problems for Windows
MANIFEST.in	Obsolete
setup.cfg	Obsolete
setup.py	Obsolete with Poetry, and has serious problems
requirements.txt	Obsolete, and has serious problems
test-requirements.txt	Obsolete, and has serious problems

CHAPTER

EIGHT

MAKING A NEW TEMPLATE

You can generate a new template. After forking Tyrannosaurus, modify the files under `tyrannosaurus/resources/`. Files are renamed according to some simple rules, described below. Note that Tyrannosaurus's own source files are deleted.

Note: The generated files are mostly trivial and are not marked as inheriting Tyrannosaurus's Apache 2.0 license. The `tests/__init__.py` file contains nontrivial code and should retain its copyright notice in the module docstring if it's kept.

`tyrannosaurus new` follows approximately these steps: 1. It clones the repo and checks out the correct tag (according to `--tyranno`). 2. It copies the correct license file from `tyrannosaurus/resources/` and deletes the others. 3. It copies all other files from `tyrannosaurus/resources`, with modified paths. Files are allowed to be overwritten. For example, `tyrannosaurus/resources/README.md` will replace Tyrannosaurus's own `README.md`. 4. It substitutes `$${...}` parameters in the files it copied.

Paths under `tyrannosuaurs/resources/` are modified according to these rules: - () `@` is a path separator (e.g. `/` on Linux). - () `$pkg` and `$project` are replaced with their values. (`$pkg` is just `$project`, lowercase, with `.`, `-`, and `_` stripped.) - () Double-extensions ending in `.txt` are fixed. For example, `.toml.txt` is changed to `.toml`. (Formally, we substitute `^.*?(\.[^.@]{1,5})\.txt$` with capture group 1.)

In addition, the `.github/` directory is copied directly, in a 1-1 file mapping. Finally, `tyrannosaurus/` is deleted.

Note: No files under `.github/` have copies under `tyrannosaurus/resources`. If you want to use parameters in template files for these files, you can make and modify copies under, for example, `tyranonosaurus/resources/.github@ISSUE_TEMPLATE@bug.md`.

Here the substitutions made in text files:

parameter	example
`$${today}`	`2021-01-06`
`$${today.year}`	`2020`
`$${today.month}`	`01`
`$${today.Month}`	`January`
`$${today.day}`	`06`
`$${now}`	`2021-01-06 14:37:03 +03:00`
`$${now.iso}`	`2021-01-06T14:37:03+03:00`
`$${now.utc}`	`2021-01-06 14:37:03 Z`
`$${now.utciso}`	`2021-01-06T14:37:03+00:00`
`$${now.hour}`	`14`

continues on next page

Table 1 – continued from previous page

parameter	example
$${now.minute}	37
$${now.second}	03
$${project}	my special-project
$${Project}	My Special-Project
$${PROJECT}	MY SPECIAL-PROJECT
$${pkg}	myspecialproject
$${Pkg}	Myspecialproject
$${license}	gpl2
$${license.family}	GPL
$${license.name}	GPL 2.0
$${license.spdx}	GPL-2.0-or-later
$${license.header}	<full copyright header>
$${license.full}	<full copyright file text>
$${version}	0.1
$${status.Name}	Alpha
$${status.name}	alpha
$${status.pypi}	3 - Alpha
$${status.dunder}	Development
$${status.Description}	An alpha state
$${status.description}	an alpha state
$${Description}	My New Project
$${description}	my new project
$${Description.}	my new project. ['.' suffix if needed]
$${authors}	Joe Johnson, Kerri Ki
$${authors.list}	["Joe Johnson", "Kerri Ki"]
$${keywords}	python, fancy
$${keywords.list}	["python", "fancy"]
$${tyranno.version}	0.9.0

CHAPTER

NINE

API REFERENCE

This page contains auto-generated API reference documentation[1].

9.1 tyrannosaurus

Metadata for Tyrannosaurus.

9.1.1 Submodules

tyrannosaurus.clean

Module that cleans up temporary files.

Module Contents

class tyrannosaurus.clean.**Clean**(*dists: bool*, *aggressive: bool*, *hard_delete: bool*, *dry_run: bool*)

clean(*path: pathlib.Path*) → collections.abc.Sequence[tuple[pathlib.Path, Optional[pathlib.Path]]]

tyrannosaurus.cli

Command-line interface.

[1] Created with sphinx-autoapi

Module Contents

tyrannosaurus.cli.**cli**

tyrannosaurus.cli.**logger**

class tyrannosaurus.cli.**CliCommands**

Commands for Tyrannosaurus.

_APACHE2

_ENV_YAML

static build(*bare: bool = flag('bare', "Don't use tox or virtualenv.")*, *dry_run: bool = flag('dry-run', "Don't run; just output. Useful for making a script template.")*, *verbose: bool = flag('verbose', 'Output more info')*) → None

Syncs, builds, and tests your project.

If `bare` is NOT set, runs:

- tyrannosaurus sync
- poetry lock
- tox
- tyrannosaurus clean

If the `bare` IS set: Runs the commands without tox and without creating a new virtualenv. This can be useful if you're using Conda and have a dependency only available through Anaconda. It's also often faster. This command is for convenience and isn't very customizable. In this case, runs:

- tyrannosaurus sync
- poetry lock
- pre-commit run check-toml
- pre-commit run check-yaml
- pre-commit run check-json
- poetry check
- poetry build
- poetry install -v
- poetry run pytest –cov
- poetry run flake8 tyrannosaurus
- poetry run flake8 docs
- poetry run flake8 –ignore=D100,D101,D102,D103,D104,S101 tests
- sphinx-build -b html docs docs/html
- tyrannosaurus clean
- pip install .

static build_internal(*bare: bool = False*, *dry: bool = False*) → collections.abc.Sequence[str]

static clean(*dists: bool = flag('dists', 'Remove dists')*, *aggressive: bool = flag('aggressive', 'Delete additional files, including .swp and .ipython_checkpoints')*, *hard_delete: bool = flag('hard-delete', 'Use shutil.rmtree instead of moving to .tyrannosaurus')*, *dry_run: bool = flag('dry-run', "Don't write; just output")*, *verbose: bool = flag('verbose', 'Output more information')*) → None

Remove unwanted files. Deletes the contents of `.tyrannosaurus`. Then trashes temporary and unwanted files and directories to a tree under `.tyrannosaurus`.

classmethod commands()

static env(*path: pathlib.Path = typer.Option(_ENV_YAML, help='Write to this path')*, *name: Optional[str] = typer.Option(None, help='Name of the environment. [default: project name]', show_default=False)*, *dev: bool = flag('dev', 'Include dev/build dependencies')*, *extras: bool = flag('extras', 'Include optional dependencies')*, *dry_run: bool = flag('dry-run', "Don't write; just output")*, *verbose: bool = flag('verbose', 'Output more info')*) → None

Generate an Anaconda environment file.

static info() → None

Print Tyrannosaurus info.

static new(*name: str = typer.Argument('project', help='The name of the project, including any dashes or capital letters')*, *license: str = typer.Option('apache2', help=f"License name. One of {', '.join((s.name for s in License))}")*, *user: Optional[str] = typer.Option(None, help='GitHub user or org')*, *authors: Optional[str] = typer.Option(None, help='Author names, comma-separated')*, *desc: str = typer.Option('A Python project', help='Short project description')*, *keywords: str = typer.Option('', help='List of <6 keywords, comma-separated', show_default=False)*, *version: str = typer.Option('0.1.0', help="Your project's semantic version")*, *status: Optional[str] = typer.Option(None, help=inspect.cleandoc("\n PyPi classifier for dev status.\n One of: planning, pre_alpha, alpha, beta, production, mature, inactive\n [default: chosen by 'version']\n "), show_choices=False)*, *track: bool = flag('track', 'Track an empty remote repo')*, *extras: bool = flag('extras', 'Include uncommon files like codemeta.json')*, *tyranno: str = typer.Option('current', help=inspect.cleandoc("\n Tyrannosaurus version to use as the template.\n Choices: an exact version, 'current' (this version), 'stable', or 'latest'.\n "))*, *prompt: bool = flag('prompt', 'Prompt for info')*, *verbose: bool = flag('verbose', 'Output more info')*) → None

Create a new project.

static recipe(*dry_run: bool = flag('dry-run', "Don't write; just output")*, *verbose: bool = flag('verbose', 'Output more info')*) → None

Generate a Conda recipe using grayskull.

static sync(*dry_run: bool = flag('dry-run', "Don't write; just output")*, *verbose: bool = flag('verbose', 'Output more info')*) → None

Sync project metadata between configured files.

static update(*auto_fix=flag('auto-fix', 'Update dependencies in place (not supported yet)', hidden=True)*, *verbose: bool = flag('verbose', 'Output more information')*) → None

Find and list dependencies that could be updated.

Parameters

- **auto_fix** – Update dependencies in place (not supported yet)
- **verbose** – Output more information

class tyrannosaurus.cli.**CliState**

dry_run :bool = False

verbose :bool = False

__post_init__()

class tyrannosaurus.cli.**Msg**

classmethod failure(*msg: str*) → None

classmethod info(*msg: str*) → None

classmethod success(*msg: str*) → None

classmethod write_info()

class tyrannosaurus.cli.**_DevNull**

Pretends to write but doesn't.

__enter__()

__exit__(*exc_type*, *exc_value*, *traceback*)

close()

flush()

write(*msg*)

tyrannosaurus.cli.**_fix_docstrings**(*commands*)

tyrannosaurus.cli.**flag**(*name: str*, *desc: str*, ***kwargs*) → typer.Option

Generates a flag-like Typer Option.

tyrannosaurus.cli.**tyranno_main**(*version: bool = flag('version', 'Write version and exit')*, *info: bool = flag('info', "Write info and exit (same as 'tyrannosaurus info')")*)

Tyrannosaurus. Tyrannosaurus can create new modern Python projects from a template and synchronize metadata across the project.

tyrannosaurus.context

Holds a "context" of metadata that was read from a pyproject.toml.

Original source: https://github.com/dmyersturnbull/tyrannosaurus Copyright 2020–2022 Douglas Myers-Turnbull Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0

Module Contents

class tyrannosaurus.context.**Context**(*path: Union[pathlib.Path, str]*, *data=None*, *dry_run: bool = False*)

back_up(*path: Union[pathlib.Path, str]*) → None

property build_sys_reqs → collections.abc.Mapping[str, str]

check_path(*path: Union[pathlib.Path, str]*) → None

delete_exact_path(*path: pathlib.Path*, *hard_delete: bool*) → tuple[Optional[pathlib.Path], Optional[pathlib.Path]]

property deps → collections.abc.Mapping[str, str]

property description → str

destroy_tmp() → bool

property dev_deps → collections.abc.Mapping[str, str]

property extras → collections.abc.Mapping[str, str]

get_bak_path(*path: Union[pathlib.Path, str]*)

has_opt(*key: str*)

has_target(*key: str*) → bool

item(*key: str*)

property license → *tyrannosaurus.enums.License*

path_source(*key: str*) → pathlib.Path

poetry(*key: str*)

property project → str

source(*key: str*)

trash(*path: str*, *hard_delete: bool*) → tuple[Optional[pathlib.Path], Optional[pathlib.Path]]

property version → str

tyrannosaurus.enums

Supporting enums.

Original source: https://github.com/dmyersturnbull/tyrannosaurus Copyright 2020–2022 Douglas Myers-Turnbull Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0

Module Contents

class tyrannosaurus.enums.**DevStatus**

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

Initialize self. See help(type(self)) for accurate signature.

alpha = alpha

beta = beta

inactive = inactive

mature = mature

planning = planning

pre_alpha = pre-alpha

production = production

property description → str

A fragment like "a production state" or "an alpha state".

property dunder → str

A string that works for `__status__`.

classmethod guess_from_version(*version: str*) → *DevStatus*

Makes a really rough guess for the status from a semantic version string.

Behavior:

```
- Guesses planning for 0.0.x (these are not semantic versions).
- Guesses alpha for pre-1.0
- Guesses production for 1.0+
```

Parameters **version** – A semantic version like "0.1.x"; can also start with 0.0.

property pypi → str

A string that is recognized as a PyPi classifier.

property true_name → str

A nice name, like "pre-alpha".

property true_value → int

1 for planning, 2 for pre-alpha, … . Same as for PyPi classifiers.

class tyrannosaurus.enums.**License**

str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

Initialize self. See help(type(self)) for accurate signature.

agpl3 = agpl3

apache2 = apache2

cc0 = cc0

ccby = ccby

ccbync = ccbync

gpl3 = gpl3

lgpl3 = lgpl3

mit = mit

mpl2 = mpl2

_read_url(*url: str*) → str

download_header() → str

download_license() → str

property family → str

property full_name → str

property header_url → str

property license_url → str

classmethod of(*value: Union[str,* License*]*) → *License*

property spdx → str

property url → str

class tyrannosaurus.enums.**Toml**(*x: collections.abc.Mapping[str, Any]*)

A thin wrapper around toml to make getting values easier.

__contains__(*items*)

__eq__(*other*)

Return self==value.

__getitem__(*items: str*)

__repr__()

Return repr(self).

__str__()

Return str(self).

get(*items: str*, *default: Any = None*) → Optional[Any]

items()

keys()

classmethod read(*path: Union[pathlib.PurePath, str]*) → *Toml*

class tyrannosaurus.enums.**TomlBuilder**

add(*key: str*, *value: Any*)

build() → *Toml*

tyrannosaurus.envs

Anaconda environments.

Module Contents

class tyrannosaurus.envs.**CondaEnv**(*name: str*, *dev: bool*, *extras: bool*)

> **_get_deps**(*context:* tyrannosaurus.context.Context) → collections.abc.Sequence[str]
>
> **create**(*context:* tyrannosaurus.context.Context, *path: pathlib.Path*) → collections.abc.Sequence[str]

tyrannosaurus.helpers

Various support code / utils.

Module Contents

class tyrannosaurus.helpers.**CondaForgeHelper**

> **has_pkg**(*name: str*)

class tyrannosaurus.helpers.**EnvHelper**

> **process**(*name: str*, *deps*, *extras: bool*) → collections.abc.Sequence[str]

class tyrannosaurus.helpers.**PyPiHelper**

> **_extract_version**(*version: str*) → Optional[str]
>
> **get_version**(*name: str*) → str
>
> **new_versions**(*pkg_versions: collections.abc.Mapping[str, str]*) → collections.abc.Mapping[str, tuple[str, str]]

class tyrannosaurus.helpers.**TrashList**(*dists: bool*, *aggressive: bool*)

> **get_list**() → collections.abc.Sequence[str]
>
> **get_patterns**() → collections.abc.Sequence[re.Pattern]
>
> **should_delete**(*p: pathlib.Path*) → bool

class tyrannosaurus.helpers.**_Env**(*authors: Optional[collections.abc.Sequence[str]]*, *user: Optional[str]*)

> **_git**(*key: str*, *name: str*) → str

tyrannosaurus.helpers.**scandir_fast**(*topdir: Union[str, pathlib.Path]*, *trash:* TrashList) → collections.abc.Sequence[pathlib.Path]

Finds paths under a dir.

Parameters

- **topdir** – The directory to search under
- **trash** – List of trash dirs

tyrannosaurus.new

Module that generates new projects.

Module Contents

class tyrannosaurus.new.**New**(*name: str*, *license_name: Union[str,* tyrannosaurus.enums.License*]*, *username: str*, *authors: collections.abc.Sequence[str]*, *description: str*, *keywords: collections.abc.Sequence[str]*, *version: str*, *status:* tyrannosaurus.enums.DevStatus, *should_track: bool*, *extras: bool*, *tyranno_vr: str*, *debug: bool = False*)

_call(*cmd: list[str]*, *cwd: Optional[pathlib.Path] = None*, *succeed: Optional[str] = None*, *fail: Union[None, str, BaseException] = None*) → Optional[str]

_checkout(*path: pathlib.Path*) → None

_checkout_rev(*path: pathlib.Path*, *tyranno_vr: str*)

_murder_evil_path_for_sure(*evil_path: pathlib.Path*) → None

There are likely to be permission issues with .git directories.

Parameters **evil_path** – The .git directory

_parse_tyranno_vr(*path: pathlib.Path*, *version: str*) → Optional[str]

_set_tyranno_vr(*path: pathlib.Path*)

_track(*path: pathlib.Path*) → None

create(*path: pathlib.Path*) → None

tyrannosaurus.parser

Parsing of various files.

Module Contents

class tyrannosaurus.parser.**LiteralParser**(*project: str*, *user: Optional[str]*, *authors: Optional[collections.abc.Sequence[str]]*, *description: str*, *keywords: collections.abc.Sequence[str]*, *version: str*, *status:* tyrannosaurus.enums.DevStatus, *license_name: Union[str,* tyrannosaurus.enums.License*]*, *tyranno_vr: str*)

_list(*v: Optional[collections.abc.Sequence[Any]]*) → str

_pretty(*v: Optional[collections.abc.Sequence[Any]]*) → str

_sentence(*v: str*) → str

download_license_template(*header: bool*) → str

parse(*s: str*) → str

tyrannosaurus.recipes

Module that generates Conda recipes and environment files.

Module Contents

class tyrannosaurus.recipes.**Recipe**(*context:* tyrannosaurus.context.Context)

create(*output_dir: Optional[pathlib.Path]*) → collections.abc.Sequence[str]

Creates the recipe file.

Parameters **output_dir** – Probably called "recipes"

tyrannosaurus.sync

Module that syncs metadata from pyproject.toml.

Module Contents

class tyrannosaurus.sync.**Sync**(*context:* tyrannosaurus.context.Context)

_careful_wrap(*s: str*, *indent: int = 4*) → str

_fix_line(*line: str*, *replace: collections.abc.Mapping[Union[str, re.Pattern], str]*) → str

_get_line_length() → int

_replace(*line: str*, *k: Union[str, re.Pattern]*, *v: str*) → Optional[str]

_replace_substrs(*path: pathlib.Path*, *replace: collections.abc.Mapping[Union[str, re.Pattern], str]*) → collections.abc.Sequence[str]

_until_line(*lines: collections.abc.Sequence[str]*, *stop_at: str*)

fix_citation() → collections.abc.Sequence[str]

fix_codemeta() → collections.abc.Sequence[str]

fix_dockerfile() → collections.abc.Sequence[str]

fix_env() → collections.abc.Sequence[str]

fix_init() → collections.abc.Sequence[str]

fix_init_internal(*init_path: pathlib.Path*) → collections.abc.Sequence[str]

fix_pyproject() → collections.abc.Sequence[str]

fix_recipe() → collections.abc.Sequence[str]

fix_recipe_internal(*recipe_path: pathlib.Path*) → collections.abc.Sequence[str]

has(*key: str*)

sync() → collections.abc.Sequence[str]

tyrannosaurus.update

For the 'update' command.

Original source: https://github.com/dmyersturnbull/tyrannosaurus Copyright 2020–2022 Douglas Myers-Turnbull Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0

Module Contents

class tyrannosaurus.update.**Update**(*context:* tyrannosaurus.context.Context)

update() → tuple[collections.abc.Mapping[str, tuple[str, str]], collections.abc.Mapping[str, tuple[str, str]]]

9.1.2 Package Contents

tyrannosaurus.**logger**

tyrannosaurus.**metadata**

tyrannosaurus.**metadata**

tyrannosaurus.**pkg**

class tyrannosaurus.**TyrannoInfo**

copyright

datestamp

now

now_utc

pretty_timestamp_utc

pretty_timestamp_with_offset

timestamp

today

version

Tyrannosaurus is an opinionated Python template for 2021 that comes with a tool to synchronize duplicate metadata across your build.

Use it to generate ready-to-go Python projects with easy testing and GitHub actions for testing and publishing. Lints on commit, tests on commit, and deploys to PyPi when you make a release on GitHub. Also see the Tyrannosaurus readme.

You can either use the command-line interface, or just clone from the GitHub source. The command-line variant will generate slightly better code by filling in your project name and options. To install, run: `pip install tyrannosaurus`

Warning: Tyrannosaurus is in an alpha build. Generally works quite well, but the `sync` command does less than advertised.

CHAPTER

TEN

WHAT'S WRONG WITH PIP OR ANACONDA?

A lot, actually. A `setup.py` can contain arbitrary code, which is a security risk, and metadata can't be extracted without installing. Pip also doesn't check for dependency conflicts. Anaconda does resolve dependencies, but some packages are not on Anaconda, and some are not kept up-to-date. It can also make for a more complex build process.

CHAPTER

ELEVEN

LIST OF INTEGRATIONS

For the full list of integrations, refer to the readme integration list.

PYTHON MODULE INDEX

t

tyrannosaurus, 21
tyrannosaurus.clean, 21
tyrannosaurus.cli, 21
tyrannosaurus.context, 24
tyrannosaurus.enums, 25
tyrannosaurus.envs, 28
tyrannosaurus.helpers, 28
tyrannosaurus.new, 29
tyrannosaurus.parser, 29
tyrannosaurus.recipes, 30
tyrannosaurus.sync, 30
tyrannosaurus.update, 31

INDEX

Symbols

_APACHE2 (*tyrannosaurus.cli.CliCommands attribute*), 22
_DevNull (*class in tyrannosaurus.cli*), 24
_ENV_YAML (*tyrannosaurus.cli.CliCommands attribute*), 22
_Env (*class in tyrannosaurus.helpers*), 28
__contains__() (*tyrannosaurus.enums.Toml method*), 27
__enter__() (*tyrannosaurus.cli._DevNull method*), 24
__eq__() (*tyrannosaurus.enums.Toml method*), 27
__exit__() (*tyrannosaurus.cli._DevNull method*), 24
__getitem__() (*tyrannosaurus.enums.Toml method*), 27
__post_init__() (*tyrannosaurus.cli.CliState method*), 24
__repr__() (*tyrannosaurus.enums.Toml method*), 27
__str__() (*tyrannosaurus.enums.Toml method*), 27
_call() (*tyrannosaurus.new.New method*), 29
_careful_wrap() (*tyrannosaurus.sync.Sync method*), 30
_checkout() (*tyrannosaurus.new.New method*), 29
_checkout_rev() (*tyrannosaurus.new.New method*), 29
_extract_version() (*tyrannosaurus.helpers.PyPiHelper method*), 28
_fix_docstrings() (*in module tyrannosaurus.cli*), 24
_fix_line() (*tyrannosaurus.sync.Sync method*), 30
_get_deps() (*tyrannosaurus.envs.CondaEnv method*), 28
_get_line_length() (*tyrannosaurus.sync.Sync method*), 30
_git() (*tyrannosaurus.helpers._Env method*), 28
_list() (*tyrannosaurus.parser.LiteralParser method*), 30
_murder_evil_path_for_sure() (*tyrannosaurus.new.New method*), 29
_parse_tyranno_vr() (*tyrannosaurus.new.New method*), 29
_pretty() (*tyrannosaurus.parser.LiteralParser method*), 30
_read_url() (*tyrannosaurus.enums.License method*), 27
_replace() (*tyrannosaurus.sync.Sync method*), 30
_replace_substrs() (*tyrannosaurus.sync.Sync method*), 31
_sentence() (*tyrannosaurus.parser.LiteralParser method*), 30
_set_tyranno_vr() (*tyrannosaurus.new.New method*), 29
_track() (*tyrannosaurus.new.New method*), 29
_until_line() (*tyrannosaurus.sync.Sync method*), 31

A

add() (*tyrannosaurus.enums.TomlBuilder method*), 27
agpl3 (*tyrannosaurus.enums.License attribute*), 26
alpha (*tyrannosaurus.enums.DevStatus attribute*), 25
apache2 (*tyrannosaurus.enums.License attribute*), 26

B

back_up() (*tyrannosaurus.context.Context method*), 24
beta (*tyrannosaurus.enums.DevStatus attribute*), 25
build() (*tyrannosaurus.cli.CliCommands static method*), 22
build() (*tyrannosaurus.enums.TomlBuilder method*), 27
build_internal() (*tyrannosaurus.cli.CliCommands static method*), 22
build_sys_reqs (*tyrannosaurus.context.Context property*), 24

C

cc0 (*tyrannosaurus.enums.License attribute*), 26
ccby (*tyrannosaurus.enums.License attribute*), 26
ccbync (*tyrannosaurus.enums.License attribute*), 26
check_path() (*tyrannosaurus.context.Context method*), 24
Clean (*class in tyrannosaurus.clean*), 21
clean() (*tyrannosaurus.clean.Clean method*), 21
clean() (*tyrannosaurus.cli.CliCommands static method*), 22
cli (*in module tyrannosaurus.cli*), 22
CliCommands (*class in tyrannosaurus.cli*), 22
CliState (*class in tyrannosaurus.cli*), 23
close() (*tyrannosaurus.cli._DevNull method*), 24

commands() (*tyrannosaurus.cli.CliCommands class method*), 23
CondaEnv (*class in tyrannosaurus.envs*), 28
CondaForgeHelper (*class in tyrannosaurus.helpers*), 28
Context (*class in tyrannosaurus.context*), 24
copyright (*tyrannosaurus.TyrannoInfo attribute*), 31
create() (*tyrannosaurus.envs.CondaEnv method*), 28
create() (*tyrannosaurus.new.New method*), 29
create() (*tyrannosaurus.recipes.Recipe method*), 30

D

datestamp (*tyrannosaurus.TyrannoInfo attribute*), 32
delete_exact_path() (*tyrannosaurus.context.Context method*), 24
deps (*tyrannosaurus.context.Context property*), 25
description (*tyrannosaurus.context.Context property*), 25
description (*tyrannosaurus.enums.DevStatus property*), 26
destroy_tmp() (*tyrannosaurus.context.Context method*), 25
dev_deps (*tyrannosaurus.context.Context property*), 25
DevStatus (*class in tyrannosaurus.enums*), 25
download_header() (*tyrannosaurus.enums.License method*), 27
download_license() (*tyrannosaurus.enums.License method*), 27
download_license_template() (*tyrannosaurus.parser.LiteralParser method*), 30
dry_run (*tyrannosaurus.cli.CliState attribute*), 23
dunder (*tyrannosaurus.enums.DevStatus property*), 26

E

env() (*tyrannosaurus.cli.CliCommands static method*), 23
EnvHelper (*class in tyrannosaurus.helpers*), 28
extras (*tyrannosaurus.context.Context property*), 25

F

failure() (*tyrannosaurus.cli.Msg class method*), 24
family (*tyrannosaurus.enums.License property*), 27
fix_citation() (*tyrannosaurus.sync.Sync method*), 31
fix_codemeta() (*tyrannosaurus.sync.Sync method*), 31
fix_dockerfile() (*tyrannosaurus.sync.Sync method*), 31
fix_env() (*tyrannosaurus.sync.Sync method*), 31
fix_init() (*tyrannosaurus.sync.Sync method*), 31
fix_init_internal() (*tyrannosaurus.sync.Sync method*), 31
fix_pyproject() (*tyrannosaurus.sync.Sync method*), 31
fix_recipe() (*tyrannosaurus.sync.Sync method*), 31
fix_recipe_internal() (*tyrannosaurus.sync.Sync method*), 31
flag() (*in module tyrannosaurus.cli*), 24
flush() (*tyrannosaurus.cli._DevNull method*), 24
full_name (*tyrannosaurus.enums.License property*), 27

G

get() (*tyrannosaurus.enums.Toml method*), 27
get_bak_path() (*tyrannosaurus.context.Context method*), 25
get_list() (*tyrannosaurus.helpers.TrashList method*), 28
get_patterns() (*tyrannosaurus.helpers.TrashList method*), 28
get_version() (*tyrannosaurus.helpers.PyPiHelper method*), 28
gpl3 (*tyrannosaurus.enums.License attribute*), 26
guess_from_version() (*tyrannosaurus.enums.DevStatus class method*), 26

H

has() (*tyrannosaurus.sync.Sync method*), 31
has_opt() (*tyrannosaurus.context.Context method*), 25
has_pkg() (*tyrannosaurus.helpers.CondaForgeHelper method*), 28
has_target() (*tyrannosaurus.context.Context method*), 25
header_url (*tyrannosaurus.enums.License property*), 27

I

inactive (*tyrannosaurus.enums.DevStatus attribute*), 26
info() (*tyrannosaurus.cli.CliCommands static method*), 23
info() (*tyrannosaurus.cli.Msg class method*), 24
item() (*tyrannosaurus.context.Context method*), 25
items() (*tyrannosaurus.enums.Toml method*), 27

K

keys() (*tyrannosaurus.enums.Toml method*), 27

L

lgpl3 (*tyrannosaurus.enums.License attribute*), 27
License (*class in tyrannosaurus.enums*), 26
license (*tyrannosaurus.context.Context property*), 25
license_url (*tyrannosaurus.enums.License property*), 27
LiteralParser (*class in tyrannosaurus.parser*), 30
logger (*in module tyrannosaurus*), 31
logger (*in module tyrannosaurus.cli*), 22

M

mature (*tyrannosaurus.enums.DevStatus attribute*), 26
metadata (*in module tyrannosaurus*), 31

mit (*tyrannosaurus.enums.License attribute*), 27
module
    tyrannosaurus, 21
    tyrannosaurus.clean, 21
    tyrannosaurus.cli, 21
    tyrannosaurus.context, 24
    tyrannosaurus.enums, 25
    tyrannosaurus.envs, 28
    tyrannosaurus.helpers, 28
    tyrannosaurus.new, 29
    tyrannosaurus.parser, 29
    tyrannosaurus.recipes, 30
    tyrannosaurus.sync, 30
    tyrannosaurus.update, 31
mpl2 (*tyrannosaurus.enums.License attribute*), 27
Msg (*class in tyrannosaurus.cli*), 24

N

New (*class in tyrannosaurus.new*), 29
new() (*tyrannosaurus.cli.CliCommands static method*), 23
new_versions() (*tyrannosaurus.helpers.PyPiHelper method*), 28
now (*tyrannosaurus.TyrannoInfo attribute*), 32
now_utc (*tyrannosaurus.TyrannoInfo attribute*), 32

O

of() (*tyrannosaurus.enums.License class method*), 27

P

parse() (*tyrannosaurus.parser.LiteralParser method*), 30
path_source() (*tyrannosaurus.context.Context method*), 25
pkg (*in module tyrannosaurus*), 31
planning (*tyrannosaurus.enums.DevStatus attribute*), 26
poetry() (*tyrannosaurus.context.Context method*), 25
pre_alpha (*tyrannosaurus.enums.DevStatus attribute*), 26
pretty_timestamp_utc (*tyrannosaurus.TyrannoInfo attribute*), 32
pretty_timestamp_with_offset (*tyrannosaurus.TyrannoInfo attribute*), 32
process() (*tyrannosaurus.helpers.EnvHelper method*), 28
production (*tyrannosaurus.enums.DevStatus attribute*), 26
project (*tyrannosaurus.context.Context property*), 25
pypi (*tyrannosaurus.enums.DevStatus property*), 26
PyPiHelper (*class in tyrannosaurus.helpers*), 28

R

read() (*tyrannosaurus.enums.Toml class method*), 27
Recipe (*class in tyrannosaurus.recipes*), 30
recipe() (*tyrannosaurus.cli.CliCommands static method*), 23

S

scandir_fast() (*in module tyrannosaurus.helpers*), 28
should_delete() (*tyrannosaurus.helpers.TrashList method*), 28
source() (*tyrannosaurus.context.Context method*), 25
spdx (*tyrannosaurus.enums.License property*), 27
success() (*tyrannosaurus.cli.Msg class method*), 24
Sync (*class in tyrannosaurus.sync*), 30
sync() (*tyrannosaurus.cli.CliCommands static method*), 23
sync() (*tyrannosaurus.sync.Sync method*), 31

T

timestamp (*tyrannosaurus.TyrannoInfo attribute*), 32
today (*tyrannosaurus.TyrannoInfo attribute*), 32
Toml (*class in tyrannosaurus.enums*), 27
TomlBuilder (*class in tyrannosaurus.enums*), 27
trash() (*tyrannosaurus.context.Context method*), 25
TrashList (*class in tyrannosaurus.helpers*), 28
true_name (*tyrannosaurus.enums.DevStatus property*), 26
true_value (*tyrannosaurus.enums.DevStatus property*), 26
tyranno_main() (*in module tyrannosaurus.cli*), 24
TyrannoInfo (*class in tyrannosaurus*), 31
tyrannosaurus
    module, 21
tyrannosaurus.clean
    module, 21
tyrannosaurus.cli
    module, 21
tyrannosaurus.context
    module, 24
tyrannosaurus.enums
    module, 25
tyrannosaurus.envs
    module, 28
tyrannosaurus.helpers
    module, 28
tyrannosaurus.new
    module, 29
tyrannosaurus.parser
    module, 29
tyrannosaurus.recipes
    module, 30
tyrannosaurus.sync
    module, 30
tyrannosaurus.update
    module, 31

U

Update (*class in tyrannosaurus.update*), 31
update() (*tyrannosaurus.cli.CliCommands static method*), 23
update() (*tyrannosaurus.update.Update method*), 31
url (*tyrannosaurus.enums.License property*), 27

V

verbose (*tyrannosaurus.cli.CliState attribute*), 24
version (*tyrannosaurus.context.Context property*), 25
version (*tyrannosaurus.TyrannoInfo attribute*), 32

W

write() (*tyrannosaurus.cli._DevNull method*), 24
write_info() (*tyrannosaurus.cli.Msg class method*), 24