mold

Release 0.2.3

Felix Hildén

Dec 07, 2021

PACKAGE

"Not the green kind."

Extensible and configurable project initialisation. We provide a command line tool that generates various types of text-based projects with simple dialogue. Mold your new projects to get up to speed quickly and confidently while following best practices. Create initialisation configurations to fit your own needs.

```
$ mold --help
$ mold python-library
```

Mold elsewhere:

- Package on PyPI
- Development on GitHub

CHAPTER

ONE

QUICK START

First, install Mold from PyPI.

```
$ pip install mold
```

Then initialise a Python package with the builtin configuration.

```
$ mold python-library
```

A series of dialogs will determine the most important information required to initialise a working library with all the necessary development tools. You might also be interested in viewing all builtin configurations or a specific configuration in more detail.

```
$ mold config list
$ mold config show python-library
```

For more details see *Command line reference* and the list of *Available plugins*. If you're new here, have a look at the *Tutorial*.

CHAPTER

TWO

MISSION

We aim to be the fastest and simplest way of creating text-based projects that have a preset structure. The trouble from starting a project to actually begin developing it should be minimised. The bulk of the work is moved to configuration that can be applied to new projects repeatedly.

Although Mold is extensible, the builtin system is opinionated and project initialisations shouldn't be considered configurable beyond specifying project metadata. Different structural or tool choices are implemented as plugins and attached to configurations for repeated initialisation.

While experienced users get value from the speedy setup, inexperienced users benefit from the preset tools even more. Less time is used when searching for ways to use the most common tools, figuring out how they are used, and debugging setup errors. Seeing new tools might even spark inspiration to learn more.

2.1 Release notes

This release notes format is based on Keep a Changelog. Mold adheres to Semantic Versioning.

2.1.1 0.2.3 (2021-12-07)

- Fix CLI example source
- Allow empty readme description

2.1.2 0.2.2 (2021-12-03)

- Update `rtd_sphinx` YAML file
- Update `github actions` CI

2.1.3 0.2.1 (2021-09-13)

- Fix `config show` crashing

2.1.4 0.2.0 (2021-09-12)

- Prefill configuration values
- Add more licenses
- Improve builtin configurations and plugins

2.1.5 0.1.0 (2021-08-16)

- Crude initial release

2.2 Command line reference

```
Extensible and configurable project initialisation.

usage: mold [configuration] [--help] [--version]
       mold <command> [arg]

COMMANDS:
[configuration]     Initialise a new project.
add [configuration] Add files to an existing project. All files
                    that a tool would write must be missing for
                    them to be added to the project.

config list         List all saved configurations.
config new [name]   Create a new configuration.
config show [name]  Show a configuration.
config del [name]   Delete a configuration.

prefill             Prefill dialog based on a configuration.
prefill show        Show prefilled dialog values.
prefill clear       Clear prefilled dialog values.

--help, -h          Display this help message and quit.
--version, -v       Display Mold version and quit.

Missing optional parameters are determined with a dialog.
```

2.3 Tutorial

2.3.1 Initialising projects

Initialising a new project is the default action of Mold. Optionally you can specify the name of the configuration to be used. If none is specified, a dialog will determine the configuration.

```
$ mold
$ mold python-project
```

A series of dialogs follows that finalise the configuration and provide project metadata. Finally the files are written to a new folder in the current working directory. See the *Command line reference* for details and more options.

2.3.2 Creating configurations

The default configurations are meant to serve as a starting point. It is possible to customise the tools applied to your projects. See the *Command line reference* for details and more options.

```
$ mold config new [config name]
```

2.3.3 Prefilling values

Some project metadata doesn't really change, like email addresses or usernames. To further reduce hassle when initialising projects, these values can be prefilled and applied to projects automatically. See the *Command line reference* for details and more options.

```
$ mold prefill
```

2.3.4 Custom extensions

Mold is built for modularity. Making custom extensions, like implementing new tools and templates, is straight forward with a builtin source code example. See *Developer reference* for more details. If the tool is something that could be applicable to many users, please open a feature request on the issue tracker!

2.4 Available plugins

Builtin Mold plugins.

2.4.1 Domains

Python

Location: `mold.plugins.domains.python`

Begin developing a python project.

Registered tools:

- python module - basic Python source module template (from mold_builtin)
- setuptools - setuptools build and dependencies for source and wheel distributions (from mold_builtin)
- pytest+tox - Pytest and linters with Tox configuration (from mold_builtin)
- github+templates - GitHub VCS host with issue templates (from mold_builtin)
- contributing Python+GitHub - contributing guide for Python projects using GitHub (from mold_builtin)
- github actions - GitHub actions with Pytest and Tox (from mold_builtin)
- minimal gitignore - git version control with minimal gitignore file (from mold_builtin)
- gitignore for Python - comprehensive gitignore file for Python (from mold_builtin)

- Apache 2.0 - permissive license preserving copyright and license notices (from mold_builtin)
- BSD 3-Clause - permissive license prohibiting use of contributor names in derived products (from mold_builtin)
- GPLv3.0 - strong copyleft license disclosing source and granting patent rights (from mold_builtin)
- MIT - permissive license only preserving copyright and license notice (from mold_builtin)
- pypi readme - basic PyPI readme file using RST (from mold_builtin)
- rst readme - basic RST readme file (from mold_builtin)
- sphinx - Sphinx documentation with initial structure and release notes (from mold_builtin)
- rtd - Read The Docs for Sphinx with badges (from mold_builtin)
- python cli - source for a Python CLI tool (from mold_builtin)
- Mold plugin - create your own Mold plugin (from mold_builtin)
- rst todo - basic RST TODO file (from mold_builtin)

2.4.2 Tools

contributing Python+GitHub

Location: `mold.plugins.tools.contributing_py_github.tool`

Contributing guide for python projects using github.

Depends on:

- pytest+tox - Pytest and linters with Tox configuration (from mold_builtin)
- github+templates - GitHub VCS host with issue templates (from mold_builtin)

github+templates

Location: `mold.plugins.tools.github.tool`

Github vcs host with issue templates.

Depends on:

- github - GitHub VCS host (from mold_builtin)

github actions

Location: `mold.plugins.tools.github_actions.tool`

Github actions with pytest and tox.

Depends on:

- github+templates - GitHub VCS host with issue templates (from mold_builtin)
- pytest+tox - Pytest and linters with Tox configuration (from mold_builtin)

minimal gitignore

Location: `mold.plugins.tools.gitignore_minimal.tool`

Git version control with minimal gitignore file.

Depends on:

- gitignore - ignore files in git version control (from mold_builtin)

gitignore for Python

Location: `mold.plugins.tools.gitignore_python.tool`

Comprehensive gitignore file for python.

Depends on:

- gitignore - ignore files in git version control (from mold_builtin)

MIT

Location: `mold.plugins.tools.license_mit.tool`

Permissive license only preserving copyright and license notice.

Depends on:

- license - license applied to the project (from mold_builtin)

pytest+tox

Location: `mold.plugins.tools.pytest_tox.tool`

Pytest and linters with tox configuration.

Depends on:

- python module - basic Python source module template (from mold_builtin)
- setuptools - setuptools build and dependencies for source and wheel distributions (from mold_builtin)

pypi readme

Location: `mold.plugins.tools.readme_pypi.tool`

Basic pypi readme file using rst.

Depends on:

- package readme - simple readme file for a package manager (from mold_builtin)
- readme - simple readme file (from mold_builtin)
- build - project build provider and dependencies (from mold_builtin)

rst readme

Location: `mold.plugins.tools.readme_rst.tool`

Basic rst readme file.

Depends on:

- readme - simple readme file (from mold_builtin)

rtd

Location: `mold.plugins.tools.rtd_sphinx.tool`

Read the docs for sphinx with badges.

Depends on:

- documentation host - provider for online documentation (from mold_builtin)
- sphinx - Sphinx documentation with initial structure and release notes (from mold_builtin)
- rst readme - basic RST readme file (from mold_builtin)
- setuptools - setuptools build and dependencies for source and wheel distributions (from mold_builtin)

setuptools

Location: `mold.plugins.tools.setuptools.tool`

Setuptools build and dependencies for source and wheel distributions.

Depends on:

- readme - simple readme file (from mold_builtin)
- build - project build provider and dependencies (from mold_builtin)
- source - project source files (from mold_builtin)
- todo - TODO file pre-filled by other tools (from mold_builtin)

python module

Location: `mold.plugins.tools.source_basic_py.tool`

Basic python source module template.

Depends on:

- source - project source files (from mold_builtin)
- readme - simple readme file (from mold_builtin)

python cli

Location: `mold.plugins.tools.source_cli_py.tool`

Source for a python cli tool.

Depends on:

- python module - basic Python source module template (from mold_builtin)
- readme - simple readme file (from mold_builtin)

Mold plugin

Location: `mold.plugins.tools.source_mold_plugin.tool`

Create your own mold plugin.

Depends on:

- python module - basic Python source module template (from mold_builtin)
- setuptools - setuptools build and dependencies for source and wheel distributions (from mold_builtin)

sphinx

Location: `mold.plugins.tools.sphinx.tool`

Sphinx documentation with initial structure and release notes.

Depends on:

- documentation - documentation engine of the project (from mold_builtin)
- readme - simple readme file (from mold_builtin)
- source - project source files (from mold_builtin)
- setuptools - setuptools build and dependencies for source and wheel distributions (from mold_builtin)
- license - license applied to the project (from mold_builtin)

rst todo

Location: `mold.plugins.tools.todo_rst.tool`

Basic rst todo file.

Depends on:

- todo - TODO file pre-filled by other tools (from mold_builtin)

2.4.3 Categories

gitignore

Location: `mold.plugins.categories.gitignore`

.gitignore file for git

license

Location: `mold.plugins.categories.license_`

License applied to the project

source

Location: `mold.plugins.categories.source`

Project source code

2.4.4 Interfaces

build

Location: `mold.plugins.face.build.interface`

Project build provider and dependencies.

Provides variables:

- `build_download_url` (<class 'str'>)
- `build_email` (<class 'str'>)
- `build_keywords` (<class 'str'>)
- `build_url` (<class 'str'>)

Accepts variables:

- `build_entry_points` (typing.Dict[str, typing.List[str]])
- `build_extra_deps` (typing.Dict[str, typing.List[str]])
- `build_project_urls` (typing.Dict[str, str])
- `build_pyproject_sections` (typing.Dict[str, typing.List[str]])
- `build_readme_file` (<class 'str'>)

Associated questions:

- `build_email`, prompt: package author email
- `build_keywords`, prompt: package keywords (space separated)

documentation

Location: `mold.plugins.face.doc.interface`

Documentation engine of the project.

Accepts variables:

- `doc_footer_lines` (`typing.List[str]`)
- `doc_header_lines` (`typing.List[str]`)
- `doc_links` (`typing.List[mold.Link]`)

Associated questions:

- `docs_semver_over_calver`, prompt: Choose a versioning scheme: Semantic Versioning (e.g. 1.7.2) or Calendar (e.g. 2018.11.03) Versioning [S]/C (leave empty for Semantic Versioning)

documentation host

Location: `mold.plugins.face.doc_host.interface`

Provider for online documentation.

Provides variables:

- `doc_host_url` (`<class 'str'>`)

github

Location: `mold.plugins.face.github.interface`

Github vcs host.

Provides variables:

- `github_repo` (`<class 'str'>`)
- `github_user` (`<class 'str'>`)

Parent interfaces:

- vcs host - online host of the version control system (from mold_builtin)

Associated questions:

- `github_user`, prompt: GitHub user name
- `github_repo`, prompt: GitHub repository (leave empty for project slug)

gitignore

Location: `mold.plugins.face.gitignore.interface`

Ignore files in git version control.

Accepts variables:

- `gitignore_items` (`typing.List[str]`)

license

Location: `mold.plugins.face.license.interface`

License applied to the project.

Provides variables:

- `license_author` (<class 'str'>)
- `license_shorthand` (<class 'str'>)
- `license_years` (<class 'str'>)

Associated questions:

- `license_author`, prompt: package author
- `license_first_year`, prompt: first year of license (leave blank for current)

package readme

Location: `mold.plugins.face.package_readme.interface`

Simple readme file for a package manager.

Accepts variables:

- `package_readme_footer_lines` (`typing.List[str]`)
- `package_readme_header_lines` (`typing.List[str]`)
- `package_readme_links` (`typing.List[mold.Link]`)

documentation host

Location: `mold.plugins.face.read_the_docs.interface`

Provider for online documentation.

Provides variables:

- `rtd_project` (<class 'str'>)

Parent interfaces:

- documentation host - provider for online documentation (from mold_builtin)

Associated questions:

- `rtd_project`, prompt: RTD project name (leave empty for project slug)

readme

Location: `mold.plugins.face.readme.interface`

Simple readme file.

Provides variables:

- `readme_description` (`<class 'str'>`)

Accepts variables:

- `readme_example_lines` (`typing.List[str]`)
- `readme_footer_lines` (`typing.List[str]`)
- `readme_header_lines` (`typing.List[str]`)
- `readme_links` (`typing.List[mold.Link]`)

Associated questions:

- `readme_description`, prompt: project description

source

Location: `mold.plugins.face.source.interface`

Project source files.

Provides variables:

- `source_full_dir` (`<class 'str'>`)
- `source_package_name` (`<class 'str'>`)
- `source_use_src_dir` (`<class 'bool'>`)

Accepts variables:

- `source_code_lines` (`typing.List[str]`)
- `source_doc_lines` (`typing.List[str]`)
- `source_import_lines` (`typing.List[str]`)

todo

Location: `mold.plugins.face.todo.interface`

Todo file pre-filled by other tools.

Accepts variables:

- `todo_items` (`typing.List[str]`)

vcs host

Location: `mold.plugins.face.vcs_host.interface`

Online host of the version control system.

Provides variables:

- `vcs_host_url` (<class 'str'>)

2.5 Developer reference

Package / repository initialisation.

See online documentation at RTD.

class mold.**Category**(*module: str*, *name: str*, *description: str*)
Collection of tools to pick one out of during initialisation.

description: str

module: str

name: str

class mold.**Domain**(*module: str*, *name: str*, *description: str*)
Project domain.

Connects all relevant tools together.

add_tool(*tool:* mold.Tool)
Register a tool to this domain.

tools: List[*mold.Tool*]

class mold.**Interface**(*module: str*, *name: str*, *description: str*, *provides: type*, *accepts: type*, *parents: List[mold.Interface] = <factory>*, *questions: List[mold.Question] = <factory>*, *post_dialog: Callable[[], None] = <factory>*)
Tool interface that provides and accepts configuration, and provides dialog.

Parameters **post_dialog** (*Callable[[], None]*) – this interface must provide the variables in "Provides" and may use the provided variables of dependencies

accepts: type

static get_namespace_dict(*namespace*) → dict
Parse variables from a namespace, i.e. provides and accepts.

parents: List[*mold.Interface*]

post_dialog: Callable[[], None]

provides: type

questions: List[*mold.Question*]

class mold.**Link**(*target: str*, *text: str*, *pre_text: Optional[str] = None*)
Tool-agnostic way of representing hyperlinks.

pre_text: str = None

target: str

text: str

class mold.**Question**(*id: str*, *prompt: str*, *prefill: bool = False*)
Question dialog.

id: str

prefill: bool = False

prompt: str

response: str

class mold.**Template**(*target_path: pathlib.Path*, *content: str*)
Jinja2 template file.

content: str

target_path: pathlib.Path

class mold.**Tool**(*module: str, name: str, description: str, depends: List[Union[mold.Tool, mold.Interface]], category: Optional[mold.Category] = None, templates: Callable[[], List[mold.Template]] = <factory>, provide_vars: Callable[[], None] = <factory>, accept_vars: Callable[[], None] = <factory>, handle_accept: Callable[[], None] = <factory>*)
Tool implementation.

Parameters

- **provide_vars** (*Callable[[], None]*) – this loader must provide the variables in "Provides", it may also modify the variables in "Accepts" or the accepted variables of other loaders
- **accept_vars** (*Callable[[], None]*) – this loader and other loaders may modify the accepted variables, this loader may modify the provided variables using the provided variables of other loaders that are depended on
- **handle_accept** (*Callable[[], None]*) – this loader may modify the accepted variables

accept_vars: Callable[[], None]

category: *mold.Category* = None

depends: List[Union[*mold.Tool*, *mold.Interface*]]

handle_accept: Callable[[], None]

provide_vars: Callable[[], None]

templates: Callable[[], List[*mold.Template*]]

mold.**templates_from_directory**(*init_file: str*) → Callable[[], List[*mold.Template*]]
Generate templates from a directory "templates" relative to path.

2.5.1 Creating plugins

Mold ships with a builtin configuration for creating plugins. Initialise a `python-library` and choose Mold plugin as the source. It contains example source code for creating each type of plugin.

```
$ mold python-library
```

2.5.2 Plugin hook

A hook for external modules to attach their domains to.

Provides the guaranteed configuration values:

- `project_name`
- `project_slug`

class mold.hook.**Accepts**
Global domain accepts variables.

class mold.hook.**Provides**
Global domain provides variables.

project_name: str = ''

project_slug: str = ''

mold.hook.**add_domain**(*domain:* mold.Domain)
Add domain to the pool of alternatives.

2.5.3 Documentation

Utilities for creating plugin documentation.

mold.doc.**render_doc**(*import_location: str*) → str
Render documentation for any Mold component.

The component is imported and introspected to determine the appropriate documentation template, which is filled with the component's attributes.

Parameters **import_location** – dotted name leading to the component, e.g. `mold.plugins.domains.python`

mold.doc.**render_docs**(*intro_text: str*, *domain_locations: Optional[List[str]] = None*, *tool_locations: Optional[List[str]] = None*, *category_locations: Optional[List[str]] = None*, *interface_locations: Optional[List[str]] = None*) → str
Render documentation for Mold components.

Components are documented with *render_doc()* and gathered by type to a single document whose main heading has a reference `plugins`.

Parameters

- **intro_text** – text to include under the main header
- **domain_locations** – domain import locations
- **tool_locations** – tool import locations
- **category_locations** – category import locations
- **interface_locations** – interface import locations

2.6 Contributing

Thank you for considering contributing to Mold! If you've found a bug or would like to propose a feature, please submit an issue.

If you'd like to get more involved, here's how. There are many valuable contributions in addition to contributing code! If you're so inclined, triaging issues, improving documentation, helping other users and reviewing existing code and PRs is equally appreciated!

The rest of this guide focuses on development and code contributions.

2.6.1 Installation

Start by cloning the most recent version, either from the main repository or a fork you created, and installing the source as an editable package. Using a virtual environment of your choice for the installation is recommended.

```
$ git clone https://github.com/felix-hilden/mold.git
$ cd mold
$ pip install -e .[dev]
```

The last command installs all the necessary extra dependencies for development.

If you forked, consider adding the upstream repository as a remote to easily update your main branch with the latest upstream changes. For tips and tricks on contributing, see how to submit a contribution, specifically opening a pull request.

2.6.2 Testing

The install can be verified, and any changes tested by running tox.

```
$ tox
```

Now tests and static checks have been run. A list of all individual tasks can be viewed with their descriptions.

```
$ tox -a -v
```

Test suite

The repository contains a suite of test cases which can be studied and run to ensure the package works as intended.

```
$ pytest
```

For tox, this is the default command when running e.g. `tox -e py`. To measure test coverage and view uncovered lines or branches run `coverage`.

```
$ coverage run
$ coverage report
```

This can be achieved with tox by running `tox -e coverage`.

Documentation

Documentation can be built locally with Sphinx.

```
$ cd docs
$ make html
```

The main page `index.html` can be found in `build/html`.

Code style

A set of style rules is followed using a variety of tools, which check code, docstrings and documentation files. To run all style checks use `tox -e lint`.

2.6.3 Releasing

Before releasing, make sure the version number is incremented and the release notes reference the new release. Running tests once more is also good practice. The following commands build source and wheel distributions to a clean directory, and publish them on PyPI according to the project name specified in the project metadata.

```
$ rm -r dist
$ python -m build
$ twine check --strict dist/*
$ twine upload dist/*
```

If you'd like to test the upload and the resulting package, use TestPyPI instead.

```
$ twine upload --repository testpypi dist/*
$ pip install --index-url https://test.pypi.org/simple/ mold
```

PYTHON MODULE INDEX

m

INDEX