{cookiecutter.app_name} Documentation

Release 0.4.0

{cookiecutter.author_name}

Feb 03, 2019

Contents:

CHAPTER 1

Plugin system overview

1.1 Conventions

For `repomate` to discover a plugin and its hooks, the following conventions need to be adhered to:

1. The PyPi package should be named `repomate-<plugin>`, where `<plugin>` is the name of the plugin.
2. The actual Python package (i.e. the directory in which the source files are located) should be called `repomate_<plugin>`. In other words, replace the hyphen in the PyPi package name with an underscore.
3. The Python module that defines the plugin's hooks/hook classes should be name `<plugin>.py`.

For an example plugin that follows these conventions, have a look at repomate-junit4. Granted that the plugin follows these conventions and is installed, it can be loaded like any other `repomate` plugin (see Using Existing Plugins).

1.2 Hooks

There are two types of hooks in `repomate`: *core hooks* and *extension hooks*.

1.2.1 Core hooks

Core hooks provide core functionality for `repomate`, and always have a default implementation in `repomate.ext.defaults`. Providing a different plugin implementation will override this behavior, thereby changing some core part of `repomate`. In general, only one implementation of a core hook will run per invocation of `repomate`. All core hooks are defined in *`repomate_plug.corehooks`*.

1.2.2 Extension hooks

Extension hooks extend the functionality of `repomate` in various ways. Unlike the core hooks, there are no default implementations of the extension hooks, and multiple implementations can be run on each invocation of `repomate`.

All extension hooks are defined in *repomate_plug.exthooks*. repomate-junit4 consists solely of extension hooks, and so do all of the repomate built-ins except for `repomate.ext.defaults`.

CHAPTER 2

Implementing hooks and writing internal plugins

Implementing a hook is fairly simple, and works the same way regardless of what type of hook it is (core or extension). If you are working with your own fork of `repomate`, all you have to do is write a small module implementing some hooks, and drop it into the `repomate.ext` sub-package (i.e. the in directory `repomate/ext` in the `repomate` repo).

There are two ways to implement hooks: as standalone functions or wrapped in a class. In the following two sections, we'll implement the *`act_on_cloned_repo()`* extension hook using both techniques. Let's call the plugin `exampleplug` and make sure it adheres to the plugin conventions.

2.1 Hook functions in a plugin class

Wrapping hook implementations in a class inheriting from *`Plugin`* is the recommended way to write plugins for `repomate`. The class does some checks to make sure that all public functions have hook function names, which comes in handy if you are in the habit of misspelling stuff (aren't we all?). Doing it this way, `exampleplug.py` would look like this:

Listing 1: exampleplug.py

```
import pathlib
import os
from typing import Union

import repomate_plug as plug

PLUGIN_NAME = 'exampleplug'

class ExamplePlugin(plug.Plugin):
    """Example plugin that implements the act_on_cloned_repo hook."""

    def act_on_cloned_repo(self,
                           path: Union[str, pathlib.Path]) -> plug.HookResult:
        """Do something with a cloned repo.
```

(continues on next page)

(continued from previous page)

```
        Args:
            path: Path to the student repo.
        Returns:
            a plug.HookResult specifying the outcome.
        """
        return plug.HookResult(
            hook=PLUGIN_NAME, status=plug.Status.WARNING, msg="This isn't quite done")
```

Dropping `exampleplug.py` into the `repomate.ext` package and running `repomate -p exampleplug clone [ADDITIONAL ARGS]` should give some not-so-interesting output from the plugin.

The name of the class really doesn't matter, it just needs to inherit from *`Plugin`*. The name of the module and hook functions matter, though. The name of the module must be the plugin name, and the hook functions must have the precise names of the hooks they implement. In fact, all public methods in a class deriving from *`Plugin`* must have names of hook functions, or the class will fail to be created. You can see that the hook returns a `HookResult`. This is used for reporting the results in `repomate`, and is entirely optional (not all hooks support it, though). Do note that if `None` is returned instead, `repomate` will not report anything for the hook. It is recommended that hooks that can return `HookResult` do. For a comprehensive example of an internal plugin implemented with a class, see the built-in javac plugin.

2.2 Standalone hook functions

Using standalone hook functions is recommended only if you don't want the safety net provided by the *`Plugin`* metaclass. It is fairly straightforward: simply mark a function with the `repomate_plug.repomate_hook` decorator. With this approach, `exampleplug.py` would look like this:

Listing 2: exampleplug.py

```
import pathlib
import os
from typing import Union

import repomate_plug as plug

PLUGIN_NAME = 'exampleplug'

@plug.repomate_hook
def act_on_cloned_repo(path: Union[str, pathlib.Path]) -> plug.HookResult:
    """Do something with a cloned repo.

    Args:
        path: Path to the student repo.
    Returns:
        a plug.HookResult specifying the outcome.
    """
    return plug.HookResult(
        hook=PLUGIN_NAME, status=plug.Status.WARNING, msg="This isn't quite done")
```

Again, dropping `exampleplug.py` into the `repomate.ext` package and running `repomate -p exampleplug clone [ADDITIONAL ARGS]` should give some not-so-interesting output from the plugin. For a more practical example of a plugin implemented using only a hook function, see the built-in pylint plugin.

CHAPTER 3

Writing external plugins (recommended and easy!)

Writing an external plugin is really easy using the repomate-plugin-cookiecutter template. First of all, you need to install cookiecutter. It's on PyPi and installs just the same as `repomate` with `pip install cookiecutter` (with whatever flags you like to use). Now, running `python3 -m cookiecutter gh:slarse/repomate-plugin-cookiecutter` will give you some prompts to answer. If you want to create a plugin called `exampleplug`, it looks something like this:

```
$ python3 -m cookiecutter gh:slarse/repomate-plugin-cookiecutter
author []: Your Name
email []: email@address.com
github_username []: your_github_username
plugin_name []: exampleplug
short_description []: An example plugin!
```

This will result in a directory called `repomate-exampleplug`, containing a fully functioning (albeit quite useless) external plugin. If you do `cd exampleplug` and then run `pip install -e .`, you will install the plugin locally. You can then use it like any of the built-in plugins, as described in Using Existing Plugins. To actually implement the behavior that you want, edit the file `repomate-exampleplug/repomate_exampleplug/exampleplug.py` to implement the hooks you want.

CHAPTER 4

repomate_plug Module Reference

4.1 API

class repomate_plug.**Plugin**
Base class for plugin classes. For plugin classes to be picked up by repomate, they must inherit from this class.

Public methods must be hook methods, i.e. implement the specification of one of the hooks defined in *PeerReviewHook* or *CloneHook*. If there are any other public methods, an error is raised on class creation. As long as the method has the correct name, it will be recognized as a hook method.

The signature of the method is not checked until the hook is registered by the repomate_plug.manager (an instance of pluggy.manager.PluginManager). Therefore, when testing a plugin, it is a good idea to include a test where it is registered with the manager to ensure that it has the correct signatures.

Private methods (i.e. methods prefixed with _) carry no such restrictions.

class repomate_plug.**HookResult**(*hook*, *status*, *msg*)

hook
Alias for field number 0

msg
Alias for field number 2

status
Alias for field number 1

class repomate_plug.**Status**
Status codes enum.

4.2 pluginmeta

class repomate_plug.pluginmeta.**Plugin**

Base class for plugin classes. For plugin classes to be picked up by `repomate`, they must inherit from this class.

Public methods must be hook methods, i.e. implement the specification of one of the hooks defined in *PeerReviewHook* or *CloneHook*. If there are any other public methods, an error is raised on class creation. As long as the method has the correct name, it will be recognized as a hook method.

The signature of the method is not checked until the hook is registered by the `repomate_plug.manager` (an instance of `pluggy.manager.PluginManager`). Therefore, when testing a plugin, it is a good idea to include a test where it is registered with the manager to ensure that it has the correct signatures.

Private methods (i.e. methods prefixed with _) carry no such restrictions.

4.3 corehooks

Hookspecs for repomate core hooks.

Core hooks provide the basic functionality of repomate. These hooks all have default implementations, but are overridden by any other implementation. All hooks in this module should have the *firstresult=True* option to the hookspec to allow for this dynamic override.

class repomate_plug.corehooks.**PeerReviewHook**

Hook functions related to allocating peer reviews.

generate_review_allocations(*master_repo_name*, *students*, *num_reviews*, *review_team_name_function*)

Generate a (peer_review_team -> reviewers) mapping for each student repository (i.e. <student>-<master_repo_name>), where len(reviewers) = num_reviews.

review_team_name_function should be used to generate review team names. It should be called like:

```
review_team_name_function(master_repo_name, student)
```

Important: There must be strictly more students than reviewers per repo (*num_reviews*). Otherwise, allocation is impossible.

Parameters

- **master_repo_name** (`str`) – Name of a master repository.
- **students** (`Iterable[str]`) – Students for which to generate peer review allocations.
- **num_reviews** (`int`) – Amount of reviews each student should perform (and consequently amount of reviewers per repo)
- **review_team_name_function** (`Callable[[str, str], str]`) – A function that takes a master repo name as its first argument, and a student username as its second, and returns a review team name.

Return type `Mapping[str, List[str]]`

Returns a (peer_review_team -> reviewers) mapping for each student repository.

4.4 exthooks

Hookspecs for repomate extension hooks.

Extension hooks add something to the functionality of repomate, but are not necessary for its operation. Currently, all extension hooks are related to cloning repos.

class repomate_plug.exthooks.**CloneHook**
Hook functions related to cloning repos.

act_on_cloned_repo(*path*, *api*)
Do something with a cloned repo.

Parameters

- **path** (Union[str, Path]) – Path to the repo.
- **api** – An instance of repomate.github_api.GitHubAPI.

Return type Optional[HookResult]

Returns optionally returns a HookResult namedtuple for reporting the outcome of the hook. May also return None, in which case no reporting will be performed for the hook.

clone_parser_hook(*clone_parser*)
Do something with the clone repos subparser before it is used used to parse CLI options. The typical task is to add options to it.

Parameters **clone_parser** (ArgumentParser) – The clone subparser.

Return type None

config_hook(*config_parser*)
Hook into the config file parsing.

Parameters **config** – the config parser after config has been read.

Return type None

parse_args(*args*)
Get the raw args from the parser. Only called for the clone parser. The typical task is to fetch any values from options added in *clone_parser_hook()*.

Parameters **args** (Namespace) – The full namespace returned by argparse.ArgumentParser.parse_args()

Return type None

4.5 exception

Exceptions for repomate_plug.

exception repomate_plug.exception.**HookNameError**
Raise when a public method in a class that inherits from *Plugin* does not have a hook name.

exception repomate_plug.exception.**PlugError**
Base class for all repomate_plug exceptions.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index