coveralls-python

Release 3.0.1

Jul 20, 2021

Contents

coveralls.io is a service for publishing your coverage stats online. This package provides seamless integration with coverage.py (and thus `py.test`, `nosetests`, etc...) in your Python projects.

CHAPTER 1

Getting Started

1.1 Usage

This package works with any CI environment. Special handling has been included for some CI service providers, but coveralls-python can run anywhere.

To get started with coveralls-python, make sure to add your repo on the coveralls.io website. If you will be using coveralls-python on TravisCI, you're done here – otherwise, take note of the "repo token" in the coveralls.io dashboard.

After that, its as simple as installing coveralls-python, collecting coverage results, and sending them to coveralls.io.

For example:

```
pip install coveralls
coverage run --source=my_package setup.py test
COVERALLS_REPO_TOKEN=tGSdG5Qcd2dcQa2oQN9GlJkL50wFZPv1j coveralls
```

coveralls-python can be configured with several environment variables, as seen above. See *Configuration* for more details.

1.2 Configuration

coveralls-python often works without any outside configuration by examining the environment it is being run in. Special handling has been added for AppVeyor, BuildKite, CircleCI, Github Actions, Jenkins, and TravisCI to make coveralls-python as close to "plug and play" as possible. It should be useable in any other CI system as well, but may need some configuration!

In cases where you do need to modify the configuration, we obey a very strict precedence order where the **latest value is used**:

- first, the CI environment will be loaded
- second, any environment variables will be loaded (eg. those which begin with `COVERALLS_`
- third, the config file is loaded (eg. `./.coveralls.yml`)

- finally, any command line flags are evaluated

Most often, you will simply need to run coveralls-python with no additional options after you have run your coverage suite:

```
coveralls
```

If you have placed your `.coveragerc` in a non-standard location, you can run:

```
coveralls --rcfile=/path/to/coveragerc
```

If you would like to override the service name (auto-discovered on most CI systems, set to `coveralls-python` otherwise):

```
coveralls --service=travis-pro
# or, via env var:
COVERALLS_SERVICE_NAME=travis-pro coveralls
```

If you are interested in merging the coverage results between multiple languages/projects, see our *multi-language* documentation.

If coveralls-python is being run on TravisCI or on GitHub Actions, it will automatically set the token for communication with coveralls.io. Otherwise, you should set the environment variable `COVERALLS_REPO_TOKEN`, which can be found on the dashboard for your project in coveralls.io:

```
COVERALLS_REPO_TOKEN=mV2Jajb8y3c6AFlcVNagHO20fiZNkXPVy coveralls
```

If you are running multiple jobs in parallel and want coveralls.io to merge those results, you should set `COVERALLS_PARALLEL` to `true` in your environment:

```
COVERALLS_PARALLEL=true coveralls
```

Later on, you can use `coveralls --finish` to let the Coveralls service know you have completed all your parallel runs:

```
coveralls --finish
```

If you are using a non-public coveralls.io instance (for example: self-hosted Coveralls Enterprise), you can set `COVERALLS_HOST` to the base URL of that insance:

```
COVERALLS_HOST="https://coveralls.aperture.com" coveralls
```

In that case, you may also be interested in disabling SSL verification:

```
COVERALLS_SKIP_SSL_VERIFY='1' coveralls
```

If you are using named jobs, you can set:

```
COVERALLS_FLAG_NAME="insert-name-here"
```

You can also set any of these values in a `.coveralls.yml` file in the root of your project repository. If you are planning to use this method, please ensure you install `coveralls[yaml]` instead of just the base `coveralls` package.

Sample `.coveralls.yml` file:

```
service_name: travis-pro
repo_token: mV2Jajb8y3c6AFlcVNagHO20fiZNkXPVy
parallel: true
coveralls_host: https://coveralls.aperture.com
```

1.2.1 Github Actions support

Coveralls natively supports jobs running on Github Actions. You can directly pass the default-provided secret GITHUB_TOKEN:

```
env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
run: |
    coveralls --service=github
```

Passing a coveralls.io token via the `COVERALLS_REPO_TOKEN` environment variable (or via the `repo_token` parameter in the config file) is not needed for Github Actions.

Sometimes Github Actions gets a little picky about the service name which needs to be used in various cases. If you run into issues, try setting the `COVERALLS_SERVICE_NAME` explicitly to either `github` or `github-actions`. It seems to be the case that you should use the `--service=github` value if you are also planning to use the `GITHUB_TOKEN` env var, and `github-actions` (which is the default) in any other case, but we've have conflicting reports on this: YMMV! See #452 for more info.

For parallel builds, you have to add a final step to let coveralls.io know the parallel build is finished:

```
jobs:
  test:
    strategy:
      matrix:
        test-name:
          - test1
          - test2
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Test
        run: ./run_tests.sh ${{ matrix.test-name }}
      - name: Upload coverage data to coveralls.io
        run: coveralls --service=github
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          COVERALLS_FLAG_NAME: ${{ matrix.test-name }}
          COVERALLS_PARALLEL: true
  coveralls:
    name: Indicate completion to coveralls.io
    needs: test
    runs-on: ubuntu-latest
    container: python:3-slim
    steps:
    - name: Finished
      run: |
        pip3 install --upgrade coveralls
        coveralls --service=github --finish
```

(continues on next page)

(continued from previous page)

```
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

The `COVERALLS_FLAG_NAME` environment variable (or the `flag_name` parameter in the config file) is optional and can be used to better identify each job on coveralls.io. It does not need to be unique across the parallel jobs.

1.2.2 Azure Pipelines support

Coveralls does not yet support Azure Pipelines, but you can make things work by impersonating another CI system such as CircleCI. For example, you can set this up by using the following script at the end of your test pipeline:

```
- script: |
    pip install coveralls
    export CIRCLE_BRANCH=$BUILD_SOURCEBRANCH
    coveralls
  displayName: 'coveralls'
  env:
    CIRCLECI: 1
    CIRCLE_BUILD_NUM: $(Build.BuildNumber)
    COVERALLS_REPO_TOKEN: $(coveralls_repo_token)
```

Note that you will also need to use the Azure Pipelines web UI to add the `coveralls_repo_token` variable to this pipeline with your repo token (which you can copy from the coveralls.io website).

As per #245, our users suggest leaving "keep this value secret" unchecked – this may be secure enough as-is, in that a user making a PR cannot access this variable.

1.3 VCS Configuration

`coveralls-python` supports `git` by default and will run the necessary `git` commands to collect the required information without any intervention.

As describe in the coveralls docs, you may also configure these values by setting environment variables. These will be used in the fallback case, eg. if `git` is not available or your project is not a `git` repository.

As described in the linked documentation, you can also use this method to support non- `git` projects:

```
GIT_ID=$(hg tip --template '{node}\n')
GIT_AUTHOR_NAME=$(hg tip --template '{author|person}\n')
GIT_AUTHOR_EMAIL=$(hg tip --template '{author|email}\n')
GIT_COMMITTER_NAME=$(hg tip --template '{author|person}\n')
GIT_COMMITTER_EMAIL=$(hg tip --template '{author|email}\n')
GIT_MESSAGE=$(hg tip --template '{desc}\n')
GIT_BRANCH=$(hg branch)
```

1.4 Usage Within Tox

Running coveralls-python from within a tox environment (v2.0 and above) requires an additional step; since coveralls-python relies on environment variables to function, you'll need to configure tox to capture those variables using the `passenv` configuration option in your `tox.ini`.

For example, on TravisCI:

```
[tox]
envlist = py34,py35,py36,py37,py38

[testenv]
passenv = TRAVIS TRAVIS_*
deps =
    coveralls
commands =
    coverage run --source=yourpackagename setup.py test
    coveralls
```

If you are configuring coveralls-python with environment variables, you should also pass those. See *Configuration* for more details.

1.4.1 AppVeyor

```
passenv = APPVEYOR APPVEYOR_*
```

All variables:

- APPVEYOR
- APPVEYOR_BUILD_ID
- APPVEYOR_REPO_BRANCH
- APPVEYOR_PULL_REQUEST_NUMBER

1.4.2 BuildKite

```
passenv = BUILDKITE BUILDKITE_*
```

All variables:

- BUILDKITE
- BUILDKITE_JOB_ID
- BUILDKITE_BRANCH

1.4.3 CircleCI

```
passenv = CIRCLECI CIRCLE_* CI_PULL_REQUEST
```

All variables:

- CIRCLECI
- CIRCLE_BUILD_NUM
- CIRCLE_BRANCH
- CI_PULL_REQUEST

1.4.4 Github Actions

```
passenv = GITHUB_*
```

All variables:

- GITHUB_ACTIONS
- GITHUB_REF
- GITHUB_SHA
- GITHUB_HEAD_REF
- GITHUB_REPOSITORY
- GITHUB_RUN_ID
- GITHUB_TOKEN

1.4.5 Jenkins

```
passenv = JENKINS_HOME BUILD_NUMBER GIT_BRANCH CI_PULL_REQUEST
```

All variables:

- JENKINS_HOME
- BUILD_NUMBER
- GIT_BRANCH
- CI_PULL_REQUEST

1.4.6 TravisCI

```
passenv = TRAVIS TRAVIS_*
```

All variables:

- TRAVIS
- TRAVIS_JOB_ID
- TRAVIS_BRANCH
- TRAVIS_PULL_REQUEST

1.4.7 SemaphoreCI

Classic

```
passenv = SEMAPHORE SEMAPHORE_EXECUTABLE_UUID SEMAPHORE_JOB_UUID SEMAPHORE_BRANCH_ID␣
↪BRANCH_NAME
```

All variables:

- SEMAPHORE

- SEMAPHORE_EXECUTABLE_UUID
- SEMAPHORE_JOB_UUID
- SEMAPHORE_BRANCH_ID
- BRANCH_NAME

2.0

```
passenv = SEMAPHORE SEMAPHORE_WORKFLOW_ID SEMAPHORE_JOB_ID SEMAPHORE_GIT_PR_NUMBER␣
↪BRANCH_NAME
```

All variables:

- SEMAPHORE
- SEMAPHORE_WORKFLOW_ID
- SEMAPHORE_JOB_ID
- SEMAPHORE_GIT_PR_NUMBER
- BRANCH_NAME

1.5 Multiple Language Support

Tracking multi-language repo coverage requires an extra setup of merging coverage data for submission.

To send coveralls.io merged data, you must use each of your coverage reporting tools in sequence, then merge the JSON data in the last step.

For example, to submit coverage for a project using both `mocha` and `py.test`, you could use the coveralls-lcov library and run:

```
# generate mocha coverage data
mocha --reporter mocha-lcov-reporter */tests/static/js/* > coverage.info

# convert data with coveralls-lcov
coveralls-lcov -v -n coverage.info > coverage.json

# merge mocha coverage with python coverage and send to coveralls
coveralls --merge=coverage.json
```

If you want to use this library to create a JSON blob for usage elsewhere, you can run:

```
coveralls --output=coverage.json
```

1.5.1 Technical Details

The JSON file to be merged must be of "coveralls-style" and contain thus a `source_files` key. The Coveralls API has more information.

1.6 Tips for .coveragerc

This section is a list of most common options for `coverage.py`, which collects all the coverage information. Coveralls is populated from this data, so it's good to know how to to configure coverage.py.

To limit the report to only your packages, specify their names (or directories):

```
[run]
source = pkgname,your_otherpackage
```

To exclude parts of your source from coverage, for example migrations folders:

```
[report]
omit = */migrations/*
```

Some lines are never executed in your tests, but that can be ok. To mark those lines use inline comments right in your source code:

```
if debug:    # pragma: no cover
    msg = "blah blah"
    log_message(msg, a)
```

Sometimes it can be tedious to mark them in code, so you can specify whole lines in .coveragerc:

```
[report]
exclude_lines =
    pragma: no cover
    def __repr__
    raise AssertionError
    raise NotImplementedError
    if __name__ == .__main__.:
```

Finally, if you're using non-default configuration file, you can specify it in the coveralls command:

```
$ coveralls --rcfile=<file>
```

1.7 Nosetests

Nosetests provide a plugin for coverage measurement of your code:

```
$ nosetests  --with-coverage --cover-package=<your_package_name>
```

However, nosetests gathers coverage for all executed code, ignoring the `source` config option in `.coveragerc`.

This well make `coveralls` report unnecessary files, which can be inconvenient. To workaround this issue, you can use the `omit` option in your `.coveragerc` to specify a list of filename patterns to leave out of reporting.

For example:

```
[report]
omit =
    */venv/*
    */my_project/ignorable_file.py
    */test_script.py
```

Note, that native `coverage.py` and `py.test` are not affected by this problem and do not require this workaround.

1.8 Troubleshooting

If you are having difficulties submitting your coverage to coveralls.io, debug mode may help you figure out the problem:

```
$ coveralls debug
```

Debug mode doesn't send anything, it just outputs prepared json and reported files list to stdout.

We also have an issue tracker on GitHub.

CHAPTER 2

About

2.1 Authors

Coveralls is written and maintained by various contributors, without whom none of this would be possible. For a full list, see GitHub.

Special thanks goes to the original maintainer, Ilya Baryshev.

CHAPTER 3

Administration

3.1 Release

This project is released on PyPI as coveralls.

To cut a new release, ensure the latest master passes all tests. Then, create a release commit:

1. Update the `CHANGELOG.md` with the new version (`clog -C CHANGELOG.md -F --setversion x.y.z`).
2. Bump the version number in `version.py`.
3. Commit and push (`git commit -am 'chore(release): bump version' && git push`)
4. Tag and push that commit with the version number (`git tag x.y.z && git push origin x.y.z`).
5. Create a new GitHub release.

To create a new PyPI release, do the following:

1. Build the sources (`python setup.py sdist bdist_wheel`).
2. Register & upload the sources. (`twine upload $PWD/dist/*`).

Then, to pin a new docker release, do the following:

1. Build the new image (`docker build --build-arg COVERALLS="coveralls==x.y.z" -t thekevjames/coveralls:x.y.z .`.
2. Push it to dockerhub (`docker push thekevjames/coveralls:x.y.z`).
3. Note: the `:latest` tag will be handled automatically by Dockerhub's automated infrastructure.

Conda should automatically create a PR on their coveralls-feedstock shortly with the updated version – if something goes wrong, the manual process would be to:

1. Fork coveralls-feedstock.
2. Update `recipe/meta.yaml` with the new version number and sha.
3. Create a PR.

4. Comment on your own PR with: "@conda-forge-admin, please rerender".
5. Merge along with the automated commit from Conda.