Building and testing Python

In this article

You can create a continuous integration (CI) workflow to build and test your Python project.

Introduction

This guide shows you how to build, test, and publish a Python package.

GitHub-hosted runners have a tools cache with pre-installed software, which includes Python and PyPy. You don't have to install anything! For a full list of up-to-date software and the pre-installed versions of Python and PyPy, see "Using GitHub-hosted runners".

Prerequisites

You should be familiar with YAML and the syntax for GitHub Actions. For more information, see "Learn GitHub Actions."

We recommend that you have a basic understanding of Python, and pip. For more information, see:

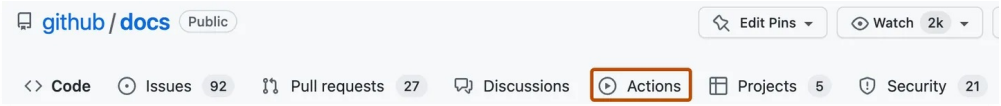
- Getting started with Python
- Pip package manager

Using a Python starter workflow

To get started quickly, add a starter workflow to the `.github/workflows` directory of your repository.

GitHub provides a starter workflow for Python that should work if your repository already contains at least one `.py` file. The subsequent sections of this guide give examples of how you can customize this starter workflow.

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.



3. If you already have a workflow in your repository, click **New workflow**.
4. The "Choose a workflow" page shows a selection of recommended starter workflows. Search for "Python application".
5. On the "Python application" workflow, click **Configure**.
6. Edit the workflow as required. For example, change the Python version.
7. Click **Commit changes**.

   The `python-app.yml` workflow file is added to the `.github/workflows` directory of your repository.

Specifying a Python version

To use a pre-installed version of Python or PyPy on a GitHub-hosted runner, use the `setup-python` action. This action finds a specific version of Python or PyPy from the tools cache on each runner and adds the necessary binaries to `PATH`, which persists for the rest of the job. If a specific version of Python is not pre-installed in the tools cache, the `setup-python` action will download and set up the appropriate version from the `python-versions` repository.

Using the `setup-python` action is the recommended way of using Python with GitHub Actions because it ensures consistent behavior across different runners and different versions of Python. If you are using a self-hosted runner, you must install Python and add it to `PATH`. For more information, see the `setup-python` action.

The table below describes the locations for the tools cache in each GitHub-hosted runner.

	Ubuntu	Mac	Windows
Tool Cache Directory	`/opt/hostedtoolcache/*`	`/Users/runner/hostedtoolcache/*`	`C:\hostedtoolcache\windows\*`
Python Tool Cache	`/opt/hostedtoolcache/Python/*`	`/Users/runner/hostedtoolcache/Python/*`	`C:\hostedtoolcache\windows\Python\*`
PyPy Tool Cache	`/opt/hostedtoolcache/PyPy/*`	`/Users/runner/hostedtoolcache/PyPy/*`	`C:\hostedtoolcache\windows\PyPy\*`

If you are using a self-hosted runner, you can configure the runner to use the `setup-python` action to manage your dependencies. For more information, see using setup-python with a self-hosted runner in the `setup-python` README.

GitHub supports semantic versioning syntax. For more information, see "Using semantic versioning" and the "Semantic versioning specification."

Using multiple Python versions

The following example uses a matrix for the job to set up multiple Python versions. For more information, see "Using a matrix for your jobs."

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["pypy3.9", "pypy3.10", "3.9", "3.10", "3.11", "3.12"]

    steps:
      - uses: actions/checkout@v4
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      # You can test your matrix by printing the current Python version
      - name: Display Python version
        run: python -c "import sys; print(sys.version)"
```

Using a specific Python version

You can configure a specific version of Python. For example, 3.10. Alternatively, you can use semantic version syntax to get the latest minor release. This example uses the latest minor release of Python 3.

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        # This is the version of the action for setting up Python, not the Python
version.
        uses: actions/setup-python@v4
        with:
          # Semantic version range syntax or exact version of a Python version
          python-version: '3.x'
          # Optional - x64 or x86 architecture, defaults to x64
          architecture: 'x64'
      # You can test your matrix by printing the current Python version
      - name: Display Python version
        run: python -c "import sys; print(sys.version)"
```

Excluding a version

If you specify a version of Python that is not available, `setup-python` fails with an error such as: `##[error]Version 3.6 with arch x64 not found`. The error message includes the available versions.

You can also use the `exclude` keyword in your workflow if there is a configuration of Python that you do not wish to run. For more information, see "Workflow syntax for GitHub Actions."

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        python-version: ["3.9", "3.10", "3.11", "pypy3.9", "pypy3.10"]
        exclude:
          - os: macos-latest
            python-version: "3.9"
          - os: windows-latest
            python-version: "3.9"
```

Using the default Python version

We recommend using `setup-python` to configure the version of Python used in your workflows because it helps make your dependencies explicit. If you don't use `setup-python`, the default version of Python set in `PATH` is used in any shell when you call `python`. The default version of Python varies between GitHub-hosted runners, which may cause unexpected changes or use an older version than expected.

GitHub-hosted runner	Description
Ubuntu	Ubuntu runners have multiple versions of system Python installed under `/usr/bin/python` and `/usr/bin/python3`. The Python versions that come packaged with Ubuntu are in addition to the versions that GitHub installs in the tools cache.
Windows	Excluding the versions of Python that are in the tools cache, Windows does not ship with an equivalent version of system Python. To maintain consistent behavior with other runners and to allow Python to be used out-of-the-box without the `setup-python` action, GitHub adds a few versions from the tools cache to `PATH`.
macOS	The macOS runners have more than one version of system Python installed, in addition to the versions that are part of the tools cache. The system Python versions are located in the `/usr/local/Cellar/python/*` directory.

Installing dependencies

GitHub-hosted runners have the pip package manager installed. You can use pip to install dependencies from the PyPI package registry before building and testing your code. For example, the YAML below installs or upgrades the `pip` package installer and the `setuptools` and `wheel` packages.

You can also cache dependencies to speed up your workflow. For more information, see "Caching dependencies to speed up workflows."

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: '3.x'
- name: Install dependencies
  run: python -m pip install --upgrade pip setuptools wheel
```

Requirements file

After you update `pip`, a typical next step is to install dependencies from *requirements.txt*. For more information, see pip.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

Caching Dependencies

You can cache and restore the dependencies using the `setup-python` action.

The following example caches dependencies for pip.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- uses: actions/setup-python@v4
  with:
    python-version: '3.11'
    cache: 'pip'
- run: pip install -r requirements.txt
- run: pip test
```

By default, the `setup-python` action searches for the dependency file (`requirements.txt` for pip, `Pipfile.lock` for pipenv or `poetry.lock` for poetry) in the whole repository. For more information, see "Caching packages dependencies" in the `setup-python` README.

If you have a custom requirement or need finer controls for caching, you can use the `cache` action. Pip caches dependencies in different locations, depending on the operating system of the runner. The path you'll need to cache may differ from the Ubuntu example above, depending on the operating system you use. For more information, see Python caching examples in the `cache` action repository.

Testing your code

You can use the same commands that you use locally to build and test your code.

Testing with pytest and pytest-cov

This example installs or upgrades `pytest` and `pytest-cov`. Tests are then run and output in JUnit format while code coverage results are output in Cobertura. For more information, see JUnit and Cobertura.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Test with pytest
  run: |
    pip install pytest pytest-cov
    pytest tests.py --doctest-modules --junitxml=junit/test-results.xml --cov=com
--cov-report=xml --cov-report=html
```

Using Ruff to lint code

The following example installs or upgrades `ruff` and uses it to lint all files. For more information, see Ruff.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Set up Python
  uses: actions/setup-python@v4
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Lint with Ruff
  run: |
    pip install ruff
    ruff --output-format=github .
  continue-on-error: true
```

The linting step has `continue-on-error: true` set. This will keep the workflow from failing if the linting step doesn't succeed. Once you've addressed all of the linting errors, you can remove this option so the workflow will catch new issues.

Running tests with tox

With GitHub Actions, you can run tests with tox and spread the work across multiple jobs. You'll need to invoke tox using the `-e py` option to choose the version of Python in your `PATH`, rather than specifying a specific version. For more information, see tox.

YAML

```yaml
name: Python package
```

```yaml
on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python: ["3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v4
      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python }}
      - name: Install tox and any other packages
        run: pip install tox
      - name: Run tox
        # Run tox using the version of Python in `PATH`
        run: tox -e py
```

Packaging workflow data as artifacts

You can upload artifacts to view after a workflow completes. For example, you may need to save log files, core dumps, test results, or screenshots. For more information, see "Storing workflow data as artifacts."

The following example demonstrates how you can use the `upload-artifact` action to archive test results from running `pytest`. For more information, see the `upload-artifact` action.

YAML

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10", "3.11"]

    steps:
      - uses: actions/checkout@v4
      - name: Setup Python # Set Python version
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      # Install pip and pytest
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install pytest
      - name: Test with pytest
        run: pytest tests.py --doctest-modules --junitxml=junit/test-results-${{
matrix.python-version }}.xml
      - name: Upload pytest test results
        uses: actions/upload-artifact@v3
        with:
          name: pytest-results-${{ matrix.python-version }}
          path: junit/test-results-${{ matrix.python-version }}.xml
```

```
        # Use always() to always run this step to publish test results when there
are test failures
        if: ${{ always() }}
```

Publishing to package registries

You can configure your workflow to publish your Python package to a package registry once your CI tests pass. This section demonstrates how you can use GitHub Actions to upload your package to PyPI each time you publish a release.

For this example, you will need to create two PyPI API tokens. You can use secrets to store the access tokens or credentials needed to publish your package. For more information, see "Using secrets in GitHub Actions."

YAML

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Upload Python Package

on:
  release:
    types: [published]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install build
      - name: Build package
        run: python -m build
      - name: Publish package
        uses: pypa/gh-action-pypi-publish@v1
        with:
          password: ${{ secrets.PYPI_API_TOKEN }}
```

For more information about the starter workflow, see `python-publish`.