pip wheel

Usage

Unix/macOS Windows

```
python -m pip wheel [options] <requirement specifier> ...
python -m pip wheel [options] -r <requirements file> ...
python -m pip wheel [options] [-e] <vcs project url> ...
python -m pip wheel [options] [-e] <local project path> ...
python -m pip wheel [options] <archive url/path> ...
```

Description

Build Wheel archives for your requirements and dependencies.

Wheel is a built-package format, and offers the advantage of not recompiling your software during every install. For more details, see the wheel docs: https://wheel.readthedocs.io/en/latest/

'pip wheel' uses the build system interface as described here: https://pip.pypa.io/en/stable/reference/build-system/

Build System Interface

This is now covered in Build System Interface.

Differences to `build`

build is a simple tool which can among other things build wheels for projects using the standard `pyproject.toml`-based build interface. It is comparable to the execution of `pip wheel --no-deps .`. It can also build source distributions which is not possible with `pip`. `pip wheel` covers the wheel scope of `build` but offers many additional features.

Options

`-w, --wheel-dir` `<dir>`

Build wheels into <dir>, where the default is the current working directory.

Skip to content

. variable: `PIP_WHEEL_DIR`)

--no-binary <format_control>

Do not use binary packages. Can be supplied multiple times, and each time adds to the existing value. Accepts either ":all:" to disable all binary packages, ":none:" to empty the set (notice the colons), or one or more package names with commas between them (no colons). Note that some packages are tricky to compile and may fail to install when this option is used on them.

(environment variable: `PIP_NO_BINARY`)

--only-binary <format_control>

Do not use source packages. Can be supplied multiple times, and each time adds to the existing value. Accepts either ":all:" to disable all source packages, ":none:" to empty the set, or one or more package names with commas between them. Packages without binary distributions will fail to install when this option is used on them.

(environment variable: `PIP_ONLY_BINARY`)

--prefer-binary

Prefer binary packages over source packages, even if the source packages are newer.

(environment variable: `PIP_PREFER_BINARY`)

--no-build-isolation

Disable isolation when building a modern source distribution. Build dependencies specified by PEP 518 must be already installed if this option is used.

(environment variable: `PIP_NO_BUILD_ISOLATION`)

--use-pep517

Use PEP 517 for building source distributions (use --no-use-pep517 to force legacy behaviour).

(environment variable: `PIP_USE_PEP517`)

--check-build-dependencies

Check the build dependencies when PEP517 is used.

(environment variable: `PIP_CHECK_BUILD_DEPENDENCIES`)

-c, --constraint <file>

Constrain versions using the given constraints file. This option can be used multiple times.

(environment variable: `PIP_CONSTRAINT`)

Skip to content

-e, --editable <path/url>

Install a project in editable mode (i.e. setuptools “develop mode”) from a local project path or a VCS url.

(environment variable: `PIP_EDITABLE`)

-r, --requirement <file>

Install from the given requirements file. This option can be used multiple times.

(environment variable: `PIP_REQUIREMENT`)

--src <dir>

Directory to check out editable projects into. The default in a virtualenv is “<venv path>/src”. The default for global installs is “<current dir>/src”.

(environment variable: `PIP_SRC`, `PIP_SOURCE`, `PIP_SOURCE_DIR`, `PIP_SOURCE_DIRECTORY`)

--ignore-requires-python

Ignore the Requires-Python information.

(environment variable: `PIP_IGNORE_REQUIRES_PYTHON`)

--no-deps

Don’t install package dependencies.

(environment variable: `PIP_NO_DEPS`, `PIP_NO_DEPENDENCIES`)

--progress-bar <progress_bar>

Specify whether the progress bar should be used [on, off, raw] (default: on)

(environment variable: `PIP_PROGRESS_BAR`)

--no-verify

Don’t verify if built wheel is valid.

(environment variable: `PIP_NO_VERIFY`)

-C, --config-settings <settings>

Configuration settings to be passed to the PEP 517 build backend. Settings take the form KEY=VALUE. Use multiple --config-settings options to pass multiple keys to the backend.

Skip to content variable: `PIP_CONFIG_SETTINGS`)

--build-option <options>

Extra arguments to be supplied to ‘setup.py bdist_wheel’.

(environment variable: `PIP_BUILD_OPTION`)

--global-option <options>

Extra global options to be supplied to the setup.py call before the install or bdist_wheel command.

(environment variable: `PIP_GLOBAL_OPTION`)

--pre

Include pre-release and development versions. By default, pip only finds stable versions.

(environment variable: `PIP_PRE`)

--require-hashes

Require a hash to check each requirement against, for repeatable installs. This option is implied when any package in a requirements file has a --hash option.

(environment variable: `PIP_REQUIRE_HASHES`)

--no-clean

Don’t clean up build directories.

(environment variable: `PIP_NO_CLEAN`)

-i, --index-url <url>

Base URL of the Python Package Index (default https://pypi.org/simple). This should point to a repository compliant with PEP 503 (the simple repository API) or a local directory laid out in the same format.

(environment variable: `PIP_INDEX_URL`, `PIP_PYPI_URL`)

--extra-index-url <url>

Extra URLs of package indexes to use in addition to --index-url. Should follow the same rules as --index-url.

(environment variable: `PIP_EXTRA_INDEX_URL`)

--no-index

Skip to content ge index (only looking at --find-links URLs instead).

(environment variable: `PIP_NO_INDEX`)

-f, --find-links <url>

If a URL or path to an html file, then parse for links to archives such as sdist (.tar.gz) or wheel (.whl) files. If a local path or file:// URL that's a directory, then look for archives in the directory listing. Links to VCS project URLs are not supported.

(environment variable: `PIP_FIND_LINKS`)

Examples

1. Build wheels for a requirement (and all its dependencies), and then install

Unix/macOS | Windows

```
python -m pip wheel --wheel-dir=/tmp/wheelhouse SomePackage
python -m pip install --no-index --find-links=/tmp/wheelhouse SomePackage
```

2. Build a wheel for a package from source

Unix/macOS | Windows

```
python -m pip wheel --no-binary SomePackage SomePackage
```