pip download

Usage

Unix/macOS Windows

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

Description

Download packages from:

- PyPI (and other indexes) using requirement specifiers.
- VCS project urls.
- Local project directories.
- Local or remote source archives.

pip also supports downloading from “requirements files”, which provide an easy way to specify a whole environment to be downloaded.

Overview

`pip download` does the same resolution and downloading as `pip install`, but instead of installing the dependencies, it collects the downloaded distributions into the directory provided (defaulting to the current directory). This directory can later be passed as the value to `pip install --find-links` to facilitate offline or locked down package installation.

Skip to content

`pip download` with the `--platform`, `--python-version`, `--implementation`, and `--abi` options provides the ability to fetch dependencies for an interpreter and system other than the ones that pip is running on. `--only-binary=:all:` or `--no-deps` is required when using any of these options. It is important to note that these options all default to the current system/interpreter, and not to the most restrictive constraints (e.g. platform any, abi none, etc). To avoid fetching dependencies that happen to match the constraint of the current interpreter (but not your target one), it is recommended to specify all of these options if you are specifying one of them. Generic dependencies (e.g. universal wheels, or dependencies with no platform, abi, or implementation constraints) will still match an over- constrained download requirement. If some of your dependencies are not available as binaries, you can build them manually for your target platform and let pip download know where to find them using `--find-links`.

Options

`-c, --constraint <file>`

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

(environment variable: `PIP_CONSTRAINT`)

`-r, --requirement <file>`

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

(environment variable: `PIP_REQUIREMENT`)

`--no-deps`

Don't install package dependencies.

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

`--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`)

`--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

Skip to content

ne or more package names with commas between them (no colons). Note that es 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`)

--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`)

--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`)

--progress-bar <progress_bar>

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

(environment variable: `PIP_PROGRESS_BAR`)

--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`)

Skip to content

--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`)

--ignore-requires-python

Ignore the Requires-Python information.

(environment variable: `PIP_IGNORE_REQUIRES_PYTHON`)

-d, --dest <dir>

Download packages into <dir>.

(environment variable: `PIP_DEST`, `PIP_DESTINATION_DIR`, `PIP_DESTINATION_DIRECTORY`)

--platform <platform>

Only use wheels compatible with <platform>. Defaults to the platform of the running system. Use this option multiple times to specify multiple platforms supported by the target interpreter.

(environment variable: `PIP_PLATFORM`)

--python-version <python_version>

The Python interpreter version to use for wheel and “Requires-Python” compatibility checks. Defaults to a version derived from the running interpreter. The version can be specified using up to three dot-separated integers (e.g. “3” for 3.0.0, “3.7” for 3.7.0, or “3.7.3”). A major-minor version can also be given as a string without dots (e.g. “37” for 3.7.0).

(environment variable: `PIP_PYTHON_VERSION`)

--implementation <implementation>

Only use wheels compatible with Python implementation <implementation>, e.g. ‘pp’, ‘jy’, ‘cp’, or ‘ip’. If not specified, then the current interpreter implementation is used. Use ‘py’ to force implementation-agnostic wheels.

(environment variable: `PIP_IMPLEMENTATION`)

Skip to content

--abi <abi>

Only use wheels compatible with Python abi <abi>, e.g. ‘pypy_41’. If not specified, then the current interpreter abi tag is used. Use this option multiple times to specify multiple abis supported by the target interpreter. Generally you will need to specify --implementation, --platform, and --python-version when using this option.

(environment variable: `PIP_ABI`)

--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

Ignore package 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`)

Skip to content

Examples

1. Download a package and all of its dependencies

Unix/macOS Windows

```
python -m pip download SomePackage
python -m pip download -d . SomePackage  # equivalent to above
python -m pip download --no-index --find-links=/tmp/wheelhouse -d /tmp/otherwheelhouse
```

2. Download a package and all of its dependencies with OSX specific interpreter constraints. This forces OSX 10.10 or lower compatibility. Since OSX deps are forward compatible, this will also match `macosx_10_9_x86_64`, `macosx_10_8_x86_64`, `macosx_10_8_intel`, etc. It will also match deps with platform `any`. Also force the interpreter version to `27` (or more generic, i.e. `2`) and implementation to `cp` (or more generic, i.e. `py`).

Unix/macOS Windows

```
python -m pip download \
   --only-binary=:all: \
   --platform macosx_10_10_x86_64 \
   --python-version 27 \
   --implementation cp \
   SomePackage
```

3. Download a package and its dependencies with linux specific constraints. Force the interpreter to be any minor version of py3k, and only accept `cp34m` or `none` as the abi.

Unix/macOS Windows

```
python -m pip download \
   --only-binary=:all: \
   --platform linux_x86_64 \
   --python-version 3 \
   --implementation cp \
   --abi cp34m \
   SomePackage
```

4. Force platform, implementation, and abi agnostic deps.

Unix/macOS Windows

Skip to content

```
python -m pip download \
   --only-binary=:all: \
   --platform any \
   --python-version 3 \
   --implementation py \
   --abi none \
   SomePackage
```

5. Even when overconstrained, this will still correctly fetch the pip universal wheel.

Unix/macOS Windows

```
$ python -m pip download \
   --only-binary=:all: \
   --platform linux_x86_64 \
   --python-version 33 \
   --implementation cp \
   --abi cp34m \
   pip>=8
```

```
$ ls pip-8.1.1-py2.py3-none-any.whl
pip-8.1.1-py2.py3-none-any.whl
```

6. Download a package supporting one of several ABIs and platforms.

   This is useful when fetching wheels for a well-defined interpreter, whose supported ABIs and platforms are known and fixed, different than the one pip is running under.

Unix/macOS Windows

```
$ python -m pip download \
   --only-binary=:all: \
   --platform manylinux1_x86_64 --platform linux_x86_64 --platform any \
   --python-version 36 \
   --implementation cp \
   --abi cp36m --abi cp36 --abi abi3 --abi none \
   SomePackage
```